Wireless bonding

Contents

- 1 Network Interface Bonding with Removable Device Support
- 2 DHCP configuration
- 3 Static Network configuration
- 4 wpa_supplicant configuration
- 5 Slave configuration
- 6 Master configuration
- 7 Enabling/Installing the Service Units
- 8 Testing the result

Related articles

- Systemd
- Network configuration
- Dnsmasq
- Software access point
- Ad-hoc networking
- Internet sharing
- Wireless network configuration
- WPA_supplicant
- Network bridge
- Netctl#Bonding
- List of applications/Internet#Network managers

Network Interface Bonding with Removable Device Support

> The Linux bonding driver provides a method for aggregating multiple network interfaces into a single logical "bonded" interface.
> **Linux Ethernet Bonding Driver HOWTO (https://www.kernel.org/doc/Documentation/networking/bonding.txt)**

The Linux kernel bonding driver can be used to provide parallel network connections to maximize throughput, or to allow redundant network connections to maximize network availability. Here is an example of using the kernel bonding driver to maximize availability, by allowing network connections to "failover" between a primary network device and any number of secondary devices, or alternatively, by selecting the highest speed connection available. This approach provides Automatic Wired and Wireless Network Configuration with Removable Device Support, using only the kernel bonding module in "active-backup" mode, the sysfs, the `iproute2 (https://www.archlinux.org/packages/?name=iproute2)` commands, and `systemd (https://www.archlinux.org/packages/?name=systemd)` "template" Unit files, **without** using systemd-networkd.

This example will run wpa_supplicant continuously on any interface, as needed, and DHCP client on a virtual "bond0" interface. This is useful, for instance, with a portable computer when you want to use the wired interface for speed and/or security when available, and the wireless interface when the wired interface is not available. The basic idea is to have two "always active" wired and wireless interfaces, then "bond" or "enslave" them to a virtual interface "master", and then let the kernel bonding module handle switching between the interfaces. Of course, this scheme can be applied to any other type of network interface, and extended to more than two physical or virtual network interfaces.

Note that host networking is managed directly with **systemd**, and that no other "connection manager" is used here, providing a more basic approach. But then also, wpa_supplicant itself can still be managed directly using `wpa_gui` from `wpa_supplicant_gui (https://aur.archlinux.org/packages/wpa_supplicant_gui/)`AUR, to scan for, select, and connect to new wireless access points/base stations.

In this example, there are six **systemd** service unit files used, along with five associated configuration files, for the **kernel bonding module (https://www.kernel.org/doc/Documentation/networking/bonding.txt)**, **wpa_supplicant**, `dhclient (https://www.archlinux.org/packages/?name=dhclient)`, and for static network configuration and specifying the primary slave network interface name. The six unit files are essentially generic service unit files which do not contain configuration data, and no modification is needed. The various service units may be stopped, started, and restarted individually without ordering errors or failed states. Any network interface device, such as typically a wired or wireless PC Card, may be removed and replaced, and reconfiguration will be automatic.

Note: All of the required systemd service files and configuration files from a working example are shown here because they are **not** the same as the standard files provided with the Arch Linux packages. Edit as required.

DHCP configuration

```
/etc/dhclient.conf
```

```
# These time-outs are aggressively short, supposing a sparsely populated network.
initial-interval 2;
reboot 5;
timeout 10;
retry 20;

# RFC 4361          Node-specific Identifiers for DHCPv4     February 2006
send dhcp-client-identifier 00:02:00:02:2e:2d:01:bd:c3:92:9a:44:2a:c4 ;
send host-name "laptop";
```

If you also run a DHCPv6 client, make sure that the DHCP Client Identifier and the DHCPv6 Client Identifier are the same DUID. The DHCP Server, dnsmasq (https://www.archlinux.org/packages/?name=dnsmasq) for instance, can be configured to give fixed IP addresses based upon multiple MAC addresses, or provided hostname, or provided Client Identifier.

```
/etc/systemd/system/dhclient@.service
```

```
[Unit]
Description= ISC dhclient on interface %I
Documentation= man:dhclient(8) man:dhclient.conf(5)

Documentation= https://www.freedesktop.org/wiki/Software/systemd/NetworkTarget/
Wants= network.target
Before= network.target
After= network-pre.target

BindsTo= sys-subsystem-net-devices-%i.device

[Service]
ExecStartPre= /usr/bin/sleep 8
ExecStart= /usr/bin/dhclient -d -pf /run/dhclient-%i -i %I

# Release the current lease and ensure that dhclient has actually stopped.
ExecStop= /usr/bin/dhclient -r -pf /run/dhclient-%i
ExecStop= /usr/bin/sleep 1

Restart= on-abnormal

[Install]
WantedBy= sys-subsystem-net-devices-%i.device
```

There is a particular issue to address. When *starting* kernel bonding, where the only working interface is the *non*-primary slave - for instance, starting with only a wireless interface available when the wired interface is the primary - then dhclient will quickly start and adopt the MAC address of the initial primary slave, and use that MAC address when attempting to communicate with the DHCP server. When the wireless interface, some short time later, is authenticated, associated, and authorized with the access point/base station, establishing a connection to the network, the bonding driver will make the wireless interface the new active interface, and change the active MAC address on the bond0 interface, to match the wireless MAC address. Because dhclient will continue to use the MAC address from the wired interface, and that MAC address is no longer accepted by the bond0 interface, all DHCP communication will fail. If there is no saved lease file in /var/lib/dhclient/dhclient.leases, then no IPv4 address will be configured, and no IPv4 traffic will be possible. It can also be seen that when dhclient starts quickly, it can read the primary slave's firmware MAC address, rather than any MAC address assigned to the device interface. If the firmware MAC address is "null", then dhclient assigns a random

MAC address. BOOTP/DHCP packets using these firmware or random MAC addresses may "succeed" in gaining a reply on the primary slave device and fail on the non-primary slave device. That can be confusing and annoying.

These are only issues with dhclient and IPv4. Fortunately, on a dhclient DHCP request, after the lease expires, dhclient "does the right thing". dhclient will function properly no matter on which slave interface it was started.

This problem **cannot** be solved by configuring the bonding driver with the default `fail_over_mac=none`. Almost all network interface devices will not pass traffic with a MAC address which is not their own. An example of this kind of warning can be seen **here (http://www.ibm.com/developerworks/linux/linux390/development_restrictions.html#net)**. Strange network behavior will be the result, where broadcast packets will pass, but ping/icmp packets will only pass in some circumstances and not others.

Ideally, dhclient would re-determine the bonding interface MAC address each time it initially retried contacting the DHCP server. Without that, a different approach is to simply delay the start of dhclient until after the kernel bonding driver has configured an active slave. If the active slave is to be the wireless interface, then wpa_supplicant will first have authenticated, associated, and authorized with the access point/base station, and dhclient will adopt the correct MAC address. If the active slave is the primary slave, again dhclient will adopt the correct MAC address. This delay is imposed with the simple `ExecStartPre= /usr/bin/sleep 8` line in the dhclient service unit file, a conservatively long delay between the time systemd starts dhclient and the supplicant and the bonding driver selects the active interface. This selection time is longest during system boot, when many processes are starting. On faster hardware, a shorter delay, perhaps `sleep 4`, may still be effective.

Static Network configuration

```
/etc/conf.d/network.conf

# These Environment names are formed by prefixing the base variable name with the interface names.
# Remember, here, these are systemd.service Environment variables, not shell variables.
# Variables that are not set or that are set to the empty string will have no effect.
# Add a list of interface names here, in the form of a comment, for reference.
# wlp2s0 enp3s0 bond0

# PRIMARY is the name of the preferred network interface in the bonded group of interfaces.
bond0PRIMARY='enp3s0'

# ADDRS is a single-quoted space separated list of IPv4 and IPv6 addresses to apply to the interface.
# The ADDRESS must be followed by a slash and a decimal number which encodes the network prefix length.
# interfaceADDRS='address1/length address2/length ...'
bond0ADDRS='192.168.0.2/24'
wlp2s0ADDRS=
enp3s0ADDRS=

# ROUTES is a single-quoted space separated list of double-quoted ip-route ROUTE specifications.
# A PREFIX must be followed by a slash and a decimal number which encodes the network prefix length.
# Remember, to be able to add a route, the host must first be able to reach the gateway.
# interfaceROUTES='"to prefix1/length] via gateway1" "to prefix2/length via gateway2" ...'
bond0ROUTES=
```

Here, for instance, a static private IPv4 address will be assigned to the bonding interface as a "fail-safe", were the DHCP server to fail or be otherwise inaccessible. The primary slave interface is also specified in this file.

```
/etc/systemd/system/static@.service

[Unit]
Description= Static Network Configuration on %I
Documentation= man:ip-address(8) man:ip-route(8) man:systemd.service(5)

Documentation= https://www.freedesktop.org/wiki/Software/systemd/NetworkTarget/
Wants= network.target
Before= network.target

BindsTo= sys-subsystem-net-devices-%i.device

[Service]
EnvironmentFile= /etc/conf.d/network.conf
```

```
Type= oneshot
RemainAfterExit= yes

# Apparently, "ip" is not synchronous/atomic, so allow some time.
ExecStart=-/usr/bin/ip link set %I up
ExecStart=-/usr/bin/sh -c '[ "$%IADDRS" ] && \
        for a in $%IADDRS;do /usr/bin/ip address add local $$a dev %I;done'
ExecStart= /usr/bin/sleep 1
ExecStart=-/usr/bin/sh -c '[ "$%IROUTES" ] && \
        for r in ${%IROUTES};do /usr/bin/ip route replace $$r dev %I;done'

ExecStop=-/usr/bin/sh -c '[ "$%IROUTES" ] && \
        for r in ${%IROUTES};do /usr/bin/ip route del $$r dev %I;done'
ExecStop= /usr/bin/sleep 1
ExecStop=-/usr/bin/sh -c '[ "$%IADDRS" ] && \
        for a in $%IADDRS;do /usr/bin/ip address del local $$a dev %I;done'

[Install]
WantedBy= sys-subsystem-net-devices-%i.device
```

Of course, static network configuration may be used as an alternative to, or in addition to, dynamic network configuration, or not at all.

wpa_supplicant configuration

```
/etc/wpa_supplicant/wpa_supplicant.conf
```

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=wheel
update_config=1
eapol_version=2
ap_scan=1
# fast_reauth=1
country=US

network={
        ssid="MyHome"
        priority=2
        proto=RSN
        group=CCMP
        pairwise=CCMP
        key_mgmt=WPA-PSK
        #psk="SuperSecret"
        psk=404fe69d94ef522ba8e7a0c456a67a583c8f39ba0b29a3ac22ebe9494cf9992b
}
```

Be careful with the actual protocol configuration in the wpa_supplicant configuration file. Using protocols incompatible with the base station can result in unstable and otherwise difficult to troubleshoot wireless connections. Pre-compute the PSK with `wpa_passphrase ssid passphrase`. `wpa_gui` can overwrite this file. Note that `wpa_supplicant` can be run on any wired or wireless interface, as needed.

```
/etc/systemd/system/supplicant@.service
```

```
[Unit]
Description= wpa_supplicant on %P
Documentation= man:wpa_supplicant(8) man:wpa_cli(8) man:wpa_supplicant.conf(5) man:wpa_passphrase(8)

Documentation= https://www.freedesktop.org/wiki/Software/systemd/NetworkTarget/
Wants= network-pre.target
Before= network-pre.target

BindsTo= sys-subsystem-net-devices-%i.device

[Service]
# Disable legacy 802.11b bitrates.
ExecStartPre=-/usr/bin/iw %I set bitrates legacy-2.4  6 9 12 18 24 36 48 54

ExecStart= /usr/bin/wpa_supplicant -c/etc/wpa_supplicant/wpa_supplicant.conf -Dnl80211,wext,wired -i %I
ExecStartPost=-/usr/bin/sh -c "/usr/bin/iw phy phy`iw dev %I info|grep wiphy|cut -d' ' -f2` set distance 10"

ExecReload= /usr/bin/wpa_cli -i %I reconfigure
ExecReload= /usr/bin/wpa_cli -i %I reassociate

ExecStop= /usr/bin/wpa_cli -i %I terminate

# Reset bitrates.
ExecStopPost=-/usr/bin/iw %I set bitrates

Restart= on-abnormal

[Install]
WantedBy= sys-subsystem-net-devices-%i.device
```

The supplicants and the DHCP client are ordered relative to the network-pre.target on shutdown. The supplicants must not be stopped before the DHCP client releases the address lease.

Remember that the `iw` commands do not work with the wired interface drivers or with older wireless drivers which rely upon the Wireless Extensions user-space driver, and will be ignored in those cases.

Slave configuration

```
/etc/systemd/system/slave@.service
```

```
[Unit]
Description= %P@%I Slave
Documentation= https://www.kernel.org/doc/Documentation/networking/bonding.txt

Documentation= https://www.freedesktop.org/wiki/Software/systemd/NetworkTarget/
Wants= network.target
Before= network.target

Requires= master@%i.service
After= master@%i.service

Before= dhclient@%i.service
Before= static@%i.service
Before= supplicant@%p.service

BindsTo= sys-subsystem-net-devices-%p.device
BindsTo= sys-subsystem-net-devices-%i.device

[Service]
Type= oneshot
RemainAfterExit= yes

ExecStart=\
 /usr/bin/ip link set %P down ;\
 /usr/bin/ip address flush dev %P ;\
 /usr/bin/ip link set %P master %I ;\
 /usr/bin/ip link set %P up

ExecStop=\
-/usr/bin/ip link set %P nomaster ;\
-/usr/bin/ip link set %P up

[Install]
WantedBy= master@%i.service
WantedBy= sys-subsystem-net-devices-%p.device
```

There is a "trick" which will be used here, in the naming of the slave service template unit files. Two environment variables are to be passed to the slave unit files, the name of the network interface, and the name of the bonding interface. Notice that there are two particular environment variables passed into a systemd unit file, %p/%P and %i/%I, these being the strings before and after the "@" character in the name of a template unit file. Here, the bonding interface name is specified in that portion of the unit file name after the "@" character, and the network interface name is passed in that portion before the "@" character. This allows two network interface names to be specified arbitrarily on the command line, without modifying the unit files themselves.

This "slave@.service" unit file will be *hard* linked to files having the same name as the network interfaces, such as "wlp2s0@.service" and "enp3s0@.service". Note that symbolic links cannot be used here, since systemd would then set %p/%P to the target file name "slave", instead of the desired network interface name.

Master configuration

/etc/modprobe.d/bonding.conf

```
# The primary slave will be configured from the systemd master unit file.

options bonding max_bonds=0 miimon=100 mode=active-backup fail_over_mac=active
```

/etc/systemd/system/master@.service

```
[Unit]
Description= %I Interface Master
Documentation= https://www.kernel.org/doc/Documentation/networking/bonding.txt

Documentation= https://www.freedesktop.org/wiki/Software/systemd/NetworkTarget/
Wants= network.target
Before= network.target

BindsTo= sys-subsystem-net-devices-%i.device
```

```
[Service]
# Environment= PRI=enp3s0
# Environment= PRI=wlp2s0
EnvironmentFile=-/etc/conf.d/network.conf

Type= oneshot
RemainAfterExit= yes

# Apparently, "ip" is not synchronous/atomic, so allow some time.
ExecStart=\
-/usr/bin/sh -c ' case %I in \
 *br*) /usr/bin/ip link add name %I type bridge ;; \
    *) /usr/bin/ip link add name %I type bond ; \
       echo -n $%IPRIMARY > /sys/devices/virtual/net/%I/bonding/primary ;; \
 esac' ;\
 /usr/bin/ip link set %I up ;\
 /usr/bin/sleep 1

ExecStop=\
 /usr/bin/ip link delete %I ;\
 /usr/bin/sleep 1

[Install]
RequiredBy= dhclient@%i.service
RequiredBy= static@%i.service
```

Of course, "Environment=" could be used here instead of the Environment file, if static network configuration is not used, and then the Environment file could be avoided. Settings from Environment files override settings made with "Environment=".

This master service unit file supports creation of a bonding master or a bridging master network interface. The type of master interface created is determined by the name of the interface. A bridging master is created when the interface name includes the character string "br", and a bonding master is created otherwise.

The RequiredBy dependencies are only here to activate the stop ordering of static or dynamic network configuration units during master stop and restart. The network configurations must be taken-down and that process completed before the slave interfaces are freed and the master interface is deleted.

Enable/Install a bonding master unit or bridging master unit *only* when the master interface is also an IP interface for the host, which is to say, when there is a static or dynamic network configuration unit Enabled/Installed on that master interface. If the bonding master or bridging master is *not* also an IP interface, then the master service unit should not be Enabled/Installed, since it will be started manually, or will be started by the slave service units, on boot, or when a network device is plugged.

Enabling/Installing the Service Units

With those preliminaries, the interface names must be specified on the command line.

Whenever a unit file is edited, afterward run:

```
# systemctl daemon-reload
```

Next, observe the available network interface names, after inserting any removable devices:

```
# ip address
```

For each interface which will be enslaved, hard link "slave@.service" to "*interface_name@*.service":

```
# ln /etc/systemd/system/slave@.service /etc/systemd/system/enp3s0@.service
# ln /etc/systemd/system/slave@.service /etc/systemd/system/wlp2s0@.service
```

Now, determine which network interface devices will need a supplicant to access the network. Typically this will just be the wireless interface. Enable/Install and start the supplicant@.service unit for each interface, as needed:

```
# systemctl --now enable supplicant@wlp2s0
```

Then, Enable/Install the slave and master units, using any desired interface name. Here, "bond0" is used:

```
# systemctl enable enp3s0@bond0 wlp2s0@bond0 master@bond0
```

Explicitly Enable/Install only the desired network configuration, specifying the interface name, here again, bond0:

```
# systemctl enable dhclient@bond0 static@bond0
```

And finally, activate the bonding interface, the DHCP client, and any static network configuration, by starting "master@bond0.service":

```
# systemctl start master@bond0
```

The master and supplicant units will be started automatically when any configured slave device appears, and in particular, when the system boots. Were any of the DHCP, or slave units to be started independently, the master unit would also be started, but normally these units will have already been started at boot.

Testing the result

Check the results:

```
# journalctl -afn100
$ ip a
$ ps wax
```

```
$ systemctl list-units '*bond*' '*master*' '*supplicant*'
```

```
UNIT                                   LOAD   ACTIVE   SUB     DESCRIPTION
sys-devices-virtual-net-bond0.device   loaded active   plugged /sys/devices/virtual/net/bond0
sys-subsystem-net-devices-bond0.device loaded active   plugged /sys/subsystem/net/devices/bond0
dhclient@bond0.service                 loaded active   running ISC dhclient on interface bond0
enp3s0@bond0.service                   loaded active   exited  enp3s0@bond0 Slave
master@bond0.service                   loaded active   exited  bond0 Interface Master
static@bond0.service                   loaded active   exited  Static Network Configuration on bond0
supplicant@enp3s0.service              loaded inactive dead    wpa_supplicant on enp3s0
supplicant@wlp2s0.service              loaded active   running wpa_supplicant on wlp2s0
wlp2s0@bond0.service                   loaded active   exited  wlp2s0@bond0 Slave
system-master.slice                    loaded active   active  system-master.slice
system-supplicant.slice                loaded active   active  system-supplicant.slice

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.

12 loaded units listed.
To show all installed unit files use 'systemctl list-unit-files'.
```

Using the wired ethernet interface,

```
$ cat /proc/net/bonding/bond0
```

```
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)

Bonding Mode: fault-tolerance (active-backup) (fail_over_mac active)
Primary Slave: enp3s0 (primary_reselect always)
Currently Active Slave: enp3s0
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0

Slave Interface: wlp2s0
MII Status: up
Speed: Unknown
Duplex: Unknown
Link Failure Count: 0
Permanent HW addr: 68:a3:c4:ac:63:d1
Slave queue ID: 0

Slave Interface: enp3s0
MII Status: up
```

```
Speed: 100 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: e8:9a:8f:2a:9e:e1
Slave queue ID: 0
```

Using the wireless interface,

```
$ cat /proc/net/bonding/bond0
```

```
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)

Bonding Mode: fault-tolerance (active-backup) (fail_over_mac active)
Primary Slave: enp3s0 (primary_reselect always)
Currently Active Slave: wlp2s0
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0

Slave Interface: wlp2s0
MII Status: up
Speed: Unknown
Duplex: Unknown
Link Failure Count: 0
Permanent HW addr: 68:a3:c4:ac:63:d1
Slave queue ID: 0

Slave Interface: enp3s0
MII Status: down
Speed: Unknown
Duplex: Unknown
Link Failure Count: 0
Permanent HW addr: e8:9a:8f:2a:9e:e1
Slave queue ID: 0
```

To tear-down the bonding interface and shutdown the master, slave, and DHCP units, simply run:

```
# systemctl stop master@bond0
```

The supplicant units can be stopped independently with

```
# systemctl stop supplicant@interface_name
```

This approach to bonded wireless networking leaves wpa_supplicant running continuously on whatever interfaces it is started. By running htop (https://www.archlinux.org/packages/?name=htop), it can be seen that wpa_supplicant, and the DHCP client daemon, seem to behave well, and do not use any noticeable CPU time.

Still, a hardware switch or rfkill can be used to actually disable the radio when desired.

Notice that the various service units are quite independent except for the ordering dependencies that have been explicitly configured. So, for instance, a dhclient configured IPv4 address may be removed without disturbing any other network configuration or functionality with

```
# systemctl stop dhclient@bond0.service
```

Similarly, an address may be released and a new address acquired with

```
# systemctl restart dhclient@bond0.service
```

And a static address or default gateway may be changed by stopping the static service unit:

```
# systemctl stop static@bond0.service
```

editing the network.conf file, and then starting the static service unit again:

```
# systemctl start static@bond0.service
```

Also, wpa_supplicant could be temporarily disabled when only the wired interface is being used, and then started again later.

This bonding interface will function properly even with only one interface available, for instance, when only a wired interface is being used. And then, simply inserting a configured wireless network card, this new wireless interface will be automatically added to the bonded interface pool, and wpa_supplicant started. Removing this wireless card again will remove the slave interface and stop wpa_supplicant.

Check that the Ethernet cable is actually plugged-in when wired networking is preferred. And use, for instance, `wpa_cli status` or `iwconfig` to verify a connection to the correct Service Set Identifier/SSID when wireless networking is used.

Retrieved from "https://wiki.archlinux.org/index.php?title=Wireless_bonding&oldid=507288"