Bond / Link Aggregation

The bonding interface provides a method for aggregating multiple network interfaces into a single logical “bonded” interface, or LAG, or ether-channel, or port-channel. The behavior of the bonded interfaces depends upon the mode; generally speaking, modes provide either hot standby or load balancing services. Additionally, link integrity monitoring may be performed.

Configuration

Address

set interfaces bonding <interface> address <address | dhcp | dhcpv6>

Configure interface *<interface>* with one or more interface addresses.

- **address** can be specified multiple times as IPv4 and/or IPv6 address, e.g. 192.0.2.1/24 and/or 2001:db8::1/64
- **dhcp** interface address is received by DHCP from a DHCP server on this segment.
- **dhcpv6** interface address is received by DHCPv6 from a DHCPv6 server on this segment.

Example:

```
set interfaces bonding bond0 address 192.0.2.1/24
set interfaces bonding bond0 address 192.0.2.2/24
set interfaces bonding bond0 address 2001:db8::ffff/64
set interfaces bonding bond0 address 2001:db8:100::ffff/64
```

set interfaces bonding <interface> ipv6 address autoconf

SLAAC RFC 4862. IPv6 hosts can configure themselves automatically when connected to an IPv6 network using the Neighbor Discovery Protocol via ICMPv6 router discovery messages. When first connected to a network, a host sends a link-local router solicitation multicast request for its configuration parameters; routers respond to such a request with a router advertisement packet that contains Internet Layer configuration parameters.

Note

This method automatically disables IPv6 traffic forwarding on the interface in question.

set interfaces bonding <interface> ipv6 address eui64 <prefix>

EUI-64 as specified in **RFC 4291** allows a host to assign iteslf a unique 64-Bit IPv6 address.

```
set interfaces bonding bond0 ipv6 address eui64 2001:db8:beef::/64
```

Link Administration

set interfaces bonding <interface> description <description>

Assign given *<description>* to interface. Description will also be passed to SNMP monitoring systems.

set interfaces bonding <interface> disable

Disable given *<interface>*. It will be placed in administratively down (`A/D`) state.

set interfaces bonding <interface> mac <mac-address>

Configure user defined MAC address on given *<interface>*.

set interfaces bonding <interface> mode <mode>

Specifies one of the bonding policies. The default is 802.3ad. Possible values are:

- **802.3ad** - IEEE 802.3ad Dynamic link aggregation. Creates aggregation groups that share the same speed and duplex settings. Utilizes all slaves in the active aggregator according to the 802.3ad specification.

  Slave selection for outgoing traffic is done according to the transmit hash policy, which may be changed from the default simple XOR policy via the `hash-policy` option, documented below.

  Note

  Not all transmit policies may be 802.3ad compliant, particularly in regards to the packet mis-ordering requirements of section 43.2.4 of the 802.3ad standard.

- **active-backup** - Active-backup policy: Only one slave in the bond is active. A different slave becomes active if, and only if, the active slave fails. The bond's MAC address is externally visible on only one port (network adapter) to avoid confusing the switch.

When a failover occurs in active-backup mode, bonding will issue one or more gratuitous ARPs on the newly active slave. One gratuitous ARP is issued for the bonding master interface and each VLAN interfaces configured above it, provided that the interface has at least one IP address configured. Gratuitous ARPs issued for VLAN interfaces are tagged with the appropriate VLAN id.

This mode provides fault tolerance. The `primary` option, documented below, affects the behavior of this mode.

- **broadcast** - Broadcast policy: transmits everything on all slave interfaces.

  This mode provides fault tolerance.
- **round-robin** - Round-robin policy: Transmit packets in sequential order from the first available slave through the last.

  This mode provides load balancing and fault tolerance.
- **transmit-load-balance** - Adaptive transmit load balancing: channel bonding that does not require any special switch support.

  Incoming traffic is received by the current slave. If the receiving slave fails, another slave takes over the MAC address of the failed receiving slave.
- **adaptive-load-balance** - Adaptive load balancing: includes transmit-load-balance plus receive load balancing for IPV4 traffic, and does not require any special switch support. The receive load balancing is achieved by ARP negotiation. The bonding driver intercepts the ARP Replies sent by the local system on their way out and overwrites the source hardware address with the unique hardware address of one of the slaves in the bond such that different peers use different hardware addresses for the server.

  Receive traffic from connections created by the server is also balanced. When the local system sends an ARP Request the bonding driver copies and saves the peer's IP information from the ARP packet. When the ARP Reply arrives from the peer, its hardware address is retrieved and the bonding driver initiates an ARP reply to this peer assigning it to one of the slaves in the bond. A problematic outcome of using ARP negotiation for balancing is that each time that an ARP request is broadcast it uses the hardware address of the bond. Hence, peers learn the hardware address of the bond and the balancing of receive traffic collapses to the current slave. This is handled by sending updates (ARP Replies) to all the peers with their individually assigned hardware address such that the traffic is redistributed. Receive traffic is also redistributed when a new slave is added to the bond and when an inactive slave is re-activated. The receive load is distributed sequentially (round robin) among the group of highest speed slaves in the bond.

  When a link is reconnected or a new slave joins the bond the receive traffic is redistributed among all active slaves in the bond by initiating ARP Replies with the selected MAC address to each of the clients. The updelay parameter (detailed below) must be set to a value equal or greater than the switch's forwarding delay so that the ARP Replies sent to the peers will not be blocked by the switch.

- **xor-hash** - XOR policy: Transmit based on the selected transmit hash policy. The default policy is a simple [(source MAC address XOR'd with destination MAC address XOR packet type ID) modulo slave count]. Alternate transmit policies may be selected via the `hash-policy` option, described below.

  This mode provides load balancing and fault tolerance.

`set interfaces bonding <interface> min-links <0-16>`

Specifies the minimum number of links that must be active before asserting carrier. It is similar to the Cisco EtherChannel min-links feature. This allows setting the minimum number of member ports that must be up (link-up state) before marking the bond device as up (carrier on). This is useful for situations where higher level services such as clustering want to ensure a minimum number of low bandwidth links are active before switchover.

This option only affects 802.3ad mode.

The default value is 0. This will cause carrier to be asserted (for 802.3ad mode) whenever there is an active aggregator, regardless of the number of available links in that aggregator.

Note

Because an aggregator cannot be active without at least one available link, setting this option to 0 or to 1 has the exact same effect.

`set interfaces bonding <interface> hash-policy <policy>`

- **layer2** - Uses XOR of hardware MAC addresses and packet type ID field to generate the hash. The formula is

  ```
  hash = source MAC XOR destination MAC XOR packet type ID
  slave number = hash modulo slave count
  ```

  This algorithm will place all traffic to a particular network peer on the same slave.

  This algorithm is 802.3ad compliant.
- **layer2+3** - This policy uses a combination of layer2 and layer3 protocol information to generate the hash. Uses XOR of hardware MAC addresses and IP addresses to generate the hash. The formula is:

```
hash = source MAC XOR destination MAC XOR packet type ID
hash = hash XOR source IP XOR destination IP
hash = hash XOR (hash RSHIFT 16)
hash = hash XOR (hash RSHIFT 8)
```

And then hash is reduced modulo slave count.

If the protocol is IPv6 then the source and destination addresses are first hashed using ipv6_addr_hash.

This algorithm will place all traffic to a particular network peer on the same slave. For non-IP traffic, the formula is the same as for the layer2 transmit hash policy.

This policy is intended to provide a more balanced distribution of traffic than layer2 alone, especially in environments where a layer3 gateway device is required to reach most destinations.

This algorithm is 802.3ad compliant.

- **layer3+4** - This policy uses upper layer protocol information, when available, to generate the hash. This allows for traffic to a particular network peer to span multiple slaves, although a single connection will not span multiple slaves.

  The formula for unfragmented TCP and UDP packets is

  ```
  hash = source port, destination port (as in the header)
  hash = hash XOR source IP XOR destination IP
  hash = hash XOR (hash RSHIFT 16)
  hash = hash XOR (hash RSHIFT 8)
  ```

  And then hash is reduced modulo slave count.

  If the protocol is IPv6 then the source and destination addresses are first hashed using ipv6_addr_hash.

  For fragmented TCP or UDP packets and all other IPv4 and IPv6 protocol traffic, the source and destination port information is omitted. For non-IP traffic, the formula is the same as for the layer2 transmit hash policy.

  This algorithm is not fully 802.3ad compliant. A single TCP or UDP conversation containing both fragmented and unfragmented packets will see packets striped across two interfaces. This may result in out of order delivery. Most traffic types will not meet this criteria, as TCP rarely fragments traffic, and most UDP traffic is not involved in extended conversations. Other implementations of 802.3ad may or may not tolerate this noncompliance.

set interfaces bonding <interface> primary <interface>

An *<interface>* specifying which slave is the primary device. The specified device will always be the active slave while it is available. Only when the primary is off-line will alternate devices be used. This is useful when one slave is preferred over another, e.g., when one slave has higher throughput than another.

The primary option is only valid for active-backup, transmit-load-balance, and adaptive-load-balance mode.

set interfaces bonding <interface> arp-monitor interval <time>

Specifies the ARP link monitoring *<time>* in seconds.

The ARP monitor works by periodically checking the slave devices to determine whether they have sent or received traffic recently (the precise criteria depends upon the bonding mode, and the state of the slave). Regular traffic is generated via ARP probes issued for the addresses specified by the `arp-monitor target` option.

If ARP monitoring is used in an etherchannel compatible mode (modes round-robin and xor-hash), the switch should be configured in a mode that evenly distributes packets across all links. If the switch is configured to distribute the packets in an XOR fashion, all replies from the ARP targets will be received on the same link which could cause the other team members to fail.

A value of 0 disables ARP monitoring. The default value is 0.

set interfaces bonding <interface> arp-monitor target <address>

Specifies the IP addresses to use as ARP monitoring peers when `arp-monitor interval` option is > 0. These are the targets of the ARP request sent to determine the health of the link to the targets.

Multiple target IP addresses can be specified. At least one IP address must be given for ARP monitoring to function.

The maximum number of targets that can be specified is 16. The default value is no IP addresses.

Member Interfaces

set interfaces bonding <interface> member interface <member>

Enslave *<member>* interface to bond *<interface>*.

Example

The following configuration on VyOS applies to all following 3rd party vendors. It creates a bond with two links and VLAN 10, 100 on the bonded interfaces with a per VIF IPv4 address.

```
# Create bonding interface bond0 with 802.3ad LACP
set interfaces bonding bond0 hash-policy 'layer2'
set interfaces bonding bond0 mode '802.3ad'

# Add the required vlans and IPv4 addresses on them
set interfaces bonding bond0 vif 10 address 192.168.0.1/24
set interfaces bonding bond0 vif 100 address 10.10.10.1/24

# Add the member interfaces to the bonding interface
set interfaces bonding bond0 member interface eth1
set interfaces bonding bond0 member interface eth2
```

Cisco Catalyst

Assign member interfaces to PortChannel

```
interface GigabitEthernet1/0/23
 description VyOS eth1
 channel-group 1 mode active
!
interface GigabitEthernet1/0/24
 description VyOS eth2
 channel-group 1 mode active
!
```

A new interface becomes present `Port-channel1`, all configuration like allowed VLAN interfaces, STP will happen here.

```
interface Port-channel1
 description LACP Channel for VyOS
 switchport trunk encapsulation dot1q
 switchport trunk allowed vlan 10,100
 switchport mode trunk
 spanning-tree portfast trunk
!
```

Juniper EX Switch

For a headstart you can use the below example on how to build a bond with two interfaces from VyOS to a Juniper EX Switch system.

```
# Create aggregated ethernet device with 802.3ad LACP and port speeds of 10gbit/s
set interfaces ae0 aggregated-ether-options link-speed 10g
set interfaces ae0 aggregated-ether-options lacp active

# Create layer 2 on the aggregated ethernet device with trunking for our vlans
set interfaces ae0 unit 0 family ethernet-switching port-mode trunk

# Add the required vlans to the device
set interfaces ae0 unit 0 family ethernet-switching vlan members 10
set interfaces ae0 unit 0 family ethernet-switching vlan members 100

# Add the two interfaces to the aggregated ethernet device, in this setup both
# ports are on the same switch (switch 0, module 1, port 0 and 1)
set interfaces xe-0/1/0 ether-options 802.3ad ae0
set interfaces xe-0/1/1 ether-options 802.3ad ae0

# But this can also be done with multiple switches in a stack, a virtual
# chassis on Juniper (switch 0 and switch 1, module 1, port 0 on both switches)
set interfaces xe-0/1/0 ether-options 802.3ad ae0
set interfaces xe-1/1/0 ether-options 802.3ad ae0
```

Aruba/HP

For a headstart you can use the below example on how to build a bond,port-channel with two interfaces from VyOS to a Aruba/HP 2510G switch.

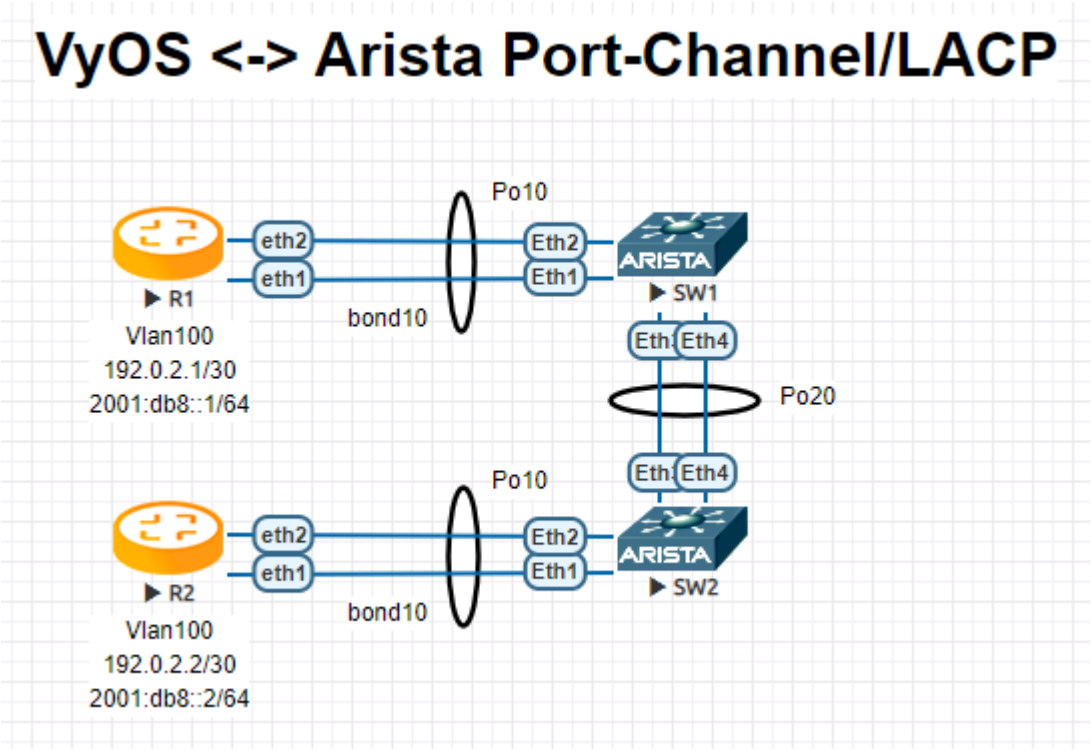
```
# Create trunk with 2 member interfaces (interface 1 and 2) and LACP
trunk 1-2 Trk1 LACP

# Add the required vlans to the trunk
vlan 10 tagged Trk1
vlan 100 tagged Trk1
```

Arista EOS

When utilizing VyOS in an environment with Arista gear you can use this blue print as an initial setup to get an LACP bond / port-channel operational between those two devices.

Lets assume the following topology:



R1

```
interfaces {
    bonding bond10 {
        hash-policy layer3+4
        member {
            interface eth1
            interface eth2
        }
        mode 802.3ad
        vif 100 {
            address 192.0.2.1/30
            address 2001:db8::1/64
        }
    }
```

R2

```
interfaces {
    bonding bond10 {
        hash-policy layer3+4
        member {
            interface eth1
            interface eth2
        }
        mode 802.3ad
        vif 100 {
            address 192.0.2.2/30
            address 2001:db8::2/64
        }
    }
```

SW1

```
!
vlan 100
   name FOO
!
interface Port-Channel10
   switchport trunk allowed vlan 100
   switchport mode trunk
   spanning-tree portfast
!
interface Port-Channel20
   switchport mode trunk
   no spanning-tree portfast auto
   spanning-tree portfast network
!
interface Ethernet1
   channel-group 10 mode active
!
interface Ethernet2
   channel-group 10 mode active
!
interface Ethernet3
   channel-group 20 mode active
!
interface Ethernet4
   channel-group 20 mode active
!
```

SW2

```
!
vlan 100
   name FOO
!
interface Port-Channel10
   switchport trunk allowed vlan 100
   switchport mode trunk
   spanning-tree portfast
!
interface Port-Channel20
   switchport mode trunk
   no spanning-tree portfast auto
   spanning-tree portfast network
!
interface Ethernet1
   channel-group 10 mode active
!
interface Ethernet2
   channel-group 10 mode active
!
interface Ethernet3
   channel-group 20 mode active
!
interface Ethernet4
   channel-group 20 mode active
!
```

Note

When using EVE-NG to lab this environment ensure you are using e1000 as the desired driver for your VyOS network interfaces. When using the regular virtio network driver no LACP PDUs will be sent by VyOS thus the port-channel will never become active!

Operation

`show interfaces bonding`

Show brief interface information.

```
vyos@vyos:~$ show interfaces bonding
Codes: S - State, L - Link, u - Up, D - Down, A - Admin Down
Interface        IP Address                        S/L  Description
---------        ----------                        ---  -----------
bond0            -                                 u/u  my-sw1 int 23 and 24
bond0.10         192.168.0.1/24                    u/u  office-net
bond0.100        10.10.10.1/24                     u/u  management-net
```

`show interfaces bonding <interface>`

Show detailed information on given *<interface>*

```
vyos@vyos:~$ show interfaces bonding bond5
bond5: <NO-CARRIER,BROADCAST,MULTICAST,MASTER,UP> mtu 1500 qdisc noqueue state DOWN
group default qlen 1000
    link/ether 00:50:56:bf:ef:aa brd ff:ff:ff:ff:ff:ff
    inet6 fe80::e862:26ff:fe72:2dac/64 scope link tentative
       valid_lft forever preferred_lft forever

    RX:  bytes  packets  errors  dropped  overrun       mcast
             0        0       0        0        0           0
    TX:  bytes  packets  errors  dropped  carrier  collisions
             0        0       0        0        0           0
```

show interfaces bonding <interface> detail

Show detailed information about the underlaying physical links on given bond *<interface>*.

```
vyos@vyos:~$ show interfaces bonding bond5 detail
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)

Bonding Mode: IEEE 802.3ad Dynamic link aggregation
Transmit Hash Policy: layer2 (0)
MII Status: down
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0

802.3ad info
LACP rate: slow
Min links: 0
Aggregator selection policy (ad_select): stable

Slave Interface: eth1
MII Status: down
Speed: Unknown
Duplex: Unknown
Link Failure Count: 0
Permanent HW addr: 00:50:56:bf:ef:aa
Slave queue ID: 0
Aggregator ID: 1
Actor Churn State: churned
Partner Churn State: churned
Actor Churned Count: 1
Partner Churned Count: 1

Slave Interface: eth2
MII Status: down
Speed: Unknown
Duplex: Unknown
Link Failure Count: 0
Permanent HW addr: 00:50:56:bf:19:26
Slave queue ID: 0
Aggregator ID: 2
Actor Churn State: churned
Partner Churn State: churned
Actor Churned Count: 1
Partner Churned Count: 1
```