Linux Ethernet Bonding Driver HOWTO

Latest update: 27 April 2011

Initial release: Thomas Davis <tadavis at lbl.gov>

Corrections, HA extensions: 2000/10/03-15:

- Willy Tarreau <willy at meta-x.org>
- Constantine Gavrilov <const-g at xpert.com>
- Chad N. Tindel <ctindel at ieee dot org>
- Janice Girouard <girouard at us dot ibm dot com>
- Jay Vosburgh <fubar at us dot ibm dot com>

Reorganized and updated Feb 2005 by Jay Vosburgh Added Sysfs information: 2006/04/24

- Mitch Williams <mitch.a.williams at intel.com>

Introduction

The Linux bonding driver provides a method for aggregating multiple network interfaces into a single logical "bonded" interface. 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.

The bonding driver originally came from Donald Becker's beowulf patches for kernel 2.0. It has changed quite a bit since, and the original tools from extreme-linux and beowulf sites will not work with this version of the driver.

For new versions of the driver, updated userspace tools, and who to ask for help, please follow the links at the end of this file.

1. Bonding Driver Installation

Most popular distro kernels ship with the bonding driver already available as a module. If your distro does not, or you have need to compile bonding from source (e.g., configuring and installing a mainline kernel from kernel.org), you'll need to perform the following steps:

1.1 Configure and build the kernel with bonding

The current version of the bonding driver is available in the drivers/net/bonding subdirectory of the most recent kernel source (which is available on http://kernel.org). Most users "rolling their own" will want to use the most recent kernel from kernel.org.

Configure kernel with "make menuconfig" (or "make xconfig" or "make config"), then select "Bonding driver support" in the "Network device support" section. It is recommended that you configure the driver as module since it is currently the only way to pass parameters to the driver or configure more than one bonding device.

Build and install the new kernel and modules.

1.2 Bonding Control Utility

It is recommended to configure bonding via iproute2 (netlink) or sysfs, the old ifenslave control utility is obsolete.

2. Bonding Driver Options

Options for the bonding driver are supplied as parameters to the bonding module at load time, or are specified via sysfs.

Module options may be given as command line arguments to the insmod or modprobe command, but are usually specified in either the `/etc/modprobe.d/*.conf` configuration files, or in a distro-specific configuration file (some of which are detailed in the next section).

Details on bonding support for sysfs is provided in the "Configuring Bonding Manually via Sysfs" section, below.

The available bonding driver parameters are listed below. If a parameter is not specified the default value is used. When initially configuring a bond, it is recommended "tail -f /var/log/messages" be run in a separate window to watch for bonding driver error messages.

It is critical that either the miimon or arp_interval and arp_ip_target parameters be specified, otherwise serious network degradation will occur during link failures. Very few devices do not support at least miimon, so there is really no reason not to use it.

Options with textual values will accept either the text name or, for backwards compatibility, the option value. E.g., "mode=802.3ad" and "mode=4" set the same mode.

The parameters are as follows:

active_slave

Specifies the new active slave for modes that support it (active-backup, balance-alb and balance-tlb). Possible values are the name of any currently enslaved interface, or an empty string. If a name is given, the slave and its link must be up in order to be selected as the new active slave. If an empty string is specified, the current active slave is cleared, and a new active slave is selected automatically.

Note that this is only available through the sysfs interface. No module parameter by this name exists.

The normal value of this option is the name of the currently active slave, or the empty string if there is no active slave or the current mode does not use an active slave.

ad_actor_sys_prio

In an AD system, this specifies the system priority. The allowed range is 1 - 65535. If the value is not specified, it takes 65535 as the default value.

This parameter has effect only in 802.3ad mode and is available through SysFs interface.

ad_actor_system

In an AD system, this specifies the mac-address for the actor in protocol packet exchanges (LACPDUs). The value cannot be a multicast address. If the all-zeroes MAC is specified, bonding will internally use the MAC of the bond itself. It is preferred to have the local-admin bit set for this mac but driver does not enforce it. If the value is not given then system defaults to using the masters' mac address as actors' system address.

This parameter has effect only in 802.3ad mode and is available through SysFs interface.

ad_select

Specifies the 802.3ad aggregation selection logic to use. The possible values and their effects are:

stable or 0

The active aggregator is chosen by largest aggregate bandwidth.

Reselection of the active aggregator occurs only when all slaves of the active aggregator are down or the active aggregator has no slaves.

This is the default value.

bandwidth or 1

The active aggregator is chosen by largest aggregate bandwidth. Reselection occurs if:

- A slave is added to or removed from the bond
- Any slave's link state changes
- Any slave's 802.3ad association state changes
- The bond's administrative state changes to up

count or 2

The active aggregator is chosen by the largest number of ports (slaves). Reselection occurs as described under the "bandwidth" setting, above.

The bandwidth and count selection policies permit failover of 802.3ad aggregations when partial failure of the active aggregator occurs. This keeps the aggregator with the highest availability (either in bandwidth or in number of ports) active at all times.

This option was added in bonding version 3.4.0.

ad_user_port_key

In an AD system, the port-key has three parts as shown below -

Bits	Use
00	Duplex
01-05	Speed
06-15	User-defined

This defines the upper 10 bits of the port key. The values can be from 0 - 1023. If not given, the system defaults to 0.

This parameter has effect only in 802.3ad mode and is available through SysFs interface.

all_slaves_active

Specifies that duplicate frames (received on inactive ports) should be dropped (0) or delivered (1).

Normally, bonding will drop duplicate frames (received on inactive ports), which is desirable for most users. But there are some times it is nice to allow duplicate frames to be delivered.

The default value is 0 (drop duplicate frames received on inactive ports).

arp_interval

Specifies the ARP link monitoring frequency in milliseconds.

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_ip_target option.

This behavior can be modified by the arp_validate option, below.

If ARP monitoring is used in an etherchannel compatible mode (modes 0 and 2), 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. ARP monitoring should not be used in conjunction with miimon. A value of 0 disables ARP monitoring. The default value is 0.

arp_ip_target

Specifies the IP addresses to use as ARP monitoring peers when arp_interval is > 0. These are the targets of the ARP request sent to determine the health of the link to the targets. Specify these values in ddd.ddd.ddd.ddd format. Multiple IP addresses must be separated by a comma. 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.

ns_ip6_target

Specifies the IPv6 addresses to use as IPv6 monitoring peers when arp_interval is > 0. These are the targets of the NS request sent to determine the health of the link to the targets. Specify these values in ffff:ffff::ffff:ffff format. Multiple IPv6 addresses must be separated by a comma. At least one IPv6 address must be given for NS/NA monitoring to function. The maximum number of targets that can be specified is 16. The default value is no IPv6 addresses.

arp_validate

Specifies whether or not ARP probes and replies should be validated in any mode that supports arp monitoring, or whether non-ARP traffic should be filtered (disregarded) for link monitoring purposes.

Possible values are:

none or 0

No validation or filtering is performed.

active or 1

Validation is performed only for the active slave.

backup or 2

Validation is performed only for backup slaves.

all or 3

Validation is performed for all slaves.

filter or 4

Filtering is applied to all slaves. No validation is performed.

filter_active or 5

Filtering is applied to all slaves, validation is performed only for the active slave.

filter_backup or 6

Filtering is applied to all slaves, validation is performed only for backup slaves.

Validation:

Enabling validation causes the ARP monitor to examine the incoming ARP requests and replies, and only consider a slave to be up if it is receiving the appropriate ARP traffic.

For an active slave, the validation checks ARP replies to confirm that they were generated by an arp_ip_target. Since backup slaves do not typically receive these replies, the validation performed for backup slaves is on the broadcast ARP request sent out via the active slave. It is possible that some switch or network configurations may result in situations wherein the backup slaves do not receive the ARP requests; in such a situation, validation of backup slaves must be disabled.

The validation of ARP requests on backup slaves is mainly helping bonding to decide which slaves are more likely to work in case of the active slave failure, it doesn't really guarantee that the backup slave will work if it's selected as the next active slave.

Validation is useful in network configurations in which multiple bonding hosts are concurrently issuing ARPs to one or more targets beyond a common switch. Should the link between the switch and target fail (but not the switch itself), the probe traffic generated by the multiple bonding instances will fool the standard ARP monitor into considering the links as still up. Use of validation can resolve this, as the ARP monitor will only consider ARP requests and replies associated with its own instance of bonding.

Filtering:

Enabling filtering causes the ARP monitor to only use incoming ARP packets for link availability purposes. Arriving packets that are not ARPs are delivered normally, but do not count when determining if a slave is available.

Filtering operates by only considering the reception of ARP packets (any ARP packet, regardless of source or destination) when determining if a slave has received traffic for link availability purposes.

Filtering is useful in network configurations in which significant levels of third party broadcast traffic would fool the standard ARP monitor into considering the links as still up. Use of filtering can resolve this, as only ARP traffic is considered for link availability purposes.

This option was added in bonding version 3.1.0.

arp_all_targets

Specifies the quantity of arp_ip_targets that must be reachable in order for the ARP monitor to consider a slave as being up. This option affects only active-backup mode for slaves with arp_validation enabled.

Possible values are:

any or 0

consider the slave up only when any of the arp_ip_targets is reachable

all or 1

consider the slave up only when all of the arp_ip_targets are reachable

arp_missed_max

Specifies the number of arp_interval monitor checks that must fail in order for an interface to be marked down by the ARP monitor.

In order to provide orderly failover semantics, backup interfaces are permitted an extra monitor check (i.e., they must fail arp_missed_max + 1 times before being marked down).

The default value is 2, and the allowable range is 1 - 255.

downdelay

Specifies the time, in milliseconds, to wait before disabling a slave after a link failure has been detected. This option is only valid for the miimon link monitor. The downdelay value should be a multiple of the miimon value; if not, it will be rounded down to the nearest multiple. The default value is 0.

fail_over_mac

Specifies whether active-backup mode should set all slaves to the same MAC address at enslavement (the traditional behavior), or, when enabled, perform special handling of the bond's MAC address in accordance with the selected policy.

Possible values are:

none or 0

This setting disables fail_over_mac, and causes bonding to set all slaves of an active-backup bond to the same

MAC address at enslavement time. This is the default.

active or 1

The "active" fail_over_mac policy indicates that the MAC address of the bond should always be the MAC address of the currently active slave. The MAC address of the slaves is not changed; instead, the MAC address of the bond changes during a failover.

This policy is useful for devices that cannot ever alter their MAC address, or for devices that refuse incoming broadcasts with their own source MAC (which interferes with the ARP monitor).

The down side of this policy is that every device on the network must be updated via gratuitous ARP, vs. just updating a switch or set of switches (which often takes place for any traffic, not just ARP traffic, if the switch snoops incoming traffic to update its tables) for the traditional method. If the gratuitous ARP is lost, communication may be disrupted.

When this policy is used in conjunction with the mii monitor, devices which assert link up prior to being able to actually transmit and receive are particularly susceptible to loss of the gratuitous ARP, and an appropriate updelay setting may be required.

follow or 2

The "follow" fail_over_mac policy causes the MAC address of the bond to be selected normally (normally the MAC address of the first slave added to the bond). However, the second and subsequent slaves are not set to this MAC address while they are in a backup role; a slave is programmed with the bond's MAC address at failover time (and the formerly active slave receives the newly active slave's MAC address).

This policy is useful for multiport devices that either become confused or incur a performance penalty when multiple ports are programmed with the same MAC address.

The default policy is none, unless the first slave cannot change its MAC address, in which case the active policy is selected by default.

This option may be modified via sysfs only when no slaves are present in the bond.

This option was added in bonding version 3.2.0. The "follow" policy was added in bonding version 3.3.0.

lacp_active

Option specifying whether to send LACPDU frames periodically.

off or 0
LACPDU frames acts as "speak when spoken to".

on or 1
LACPDU frames are sent along the configured links periodically. See lacp_rate for more details.

The default is on.

lacp_rate

Option specifying the rate in which we'll ask our link partner to transmit LACPDU packets in 802.3ad mode. Possible values are:

slow or 0
Request partner to transmit LACPDUs every 30 seconds

fast or 1
Request partner to transmit LACPDUs every 1 second

The default is slow.

max_bonds

Specifies the number of bonding devices to create for this instance of the bonding driver. E.g., if max_bonds is 3, and the bonding driver is not already loaded, then bond0, bond1 and bond2 will be created. The default value is 1. Specifying a value of 0 will load bonding, but will not create any devices.

miimon

Specifies the MII link monitoring frequency in milliseconds. This determines how often the link state of each slave is inspected for link failures. A value of zero disables MII link monitoring. A value of 100 is a good starting point. The use_carrier option, below, affects how the link state is determined. See the High Availability section for additional information. The default value is 0.

min_links

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 affect 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 that, 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.

mode

Specifies one of the bonding policies. The default is balance-rr (round robin). Possible values are:

balance-rr or 0

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

active-backup or 1

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.

In bonding version 2.6.2 or later, 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.

balance-xor or 2

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 xmit_hash_policy option, described below.

This mode provides load balancing and fault tolerance.

broadcast or 3

Broadcast policy: transmits everything on all slave interfaces. This mode provides fault tolerance.

802.3ad or 4

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 xmit_hash_policy option, documented below. Note that 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. Differing peer implementations will have varying tolerances for noncompliance.

Prerequisites:

1. Ethtool support in the base drivers for retrieving the speed and duplex of each slave.

2. A switch that supports IEEE 802.3ad Dynamic link aggregation.

Most switches will require some type of configuration to enable 802.3ad mode.

balance-tlb or 5

Adaptive transmit load balancing: channel bonding that does not require any special switch support.

In tlb_dynamic_lb=1 mode; the outgoing traffic is distributed according to the current load (computed relative to the speed) on each slave.

In tlb_dynamic_lb=0 mode; the load balancing based on current load is disabled and the load is distributed only using the hash distribution.

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.

Prerequisite:

Ethtool support in the base drivers for retrieving the speed of each slave.

balance-alb or 6

Adaptive load balancing: includes balance-tlb plus receive load balancing (rlb) 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.

Prerequisites:

1. Ethtool support in the base drivers for retrieving the speed of each slave.

2. Base driver support for setting the hardware address of a device while it is open. This is required so that there will always be one slave in the team using the bond hardware address (the curr_active_slave) while having a unique hardware address for each slave in the bond. If the curr_active_slave fails its hardware address is swapped with the new curr_active_slave that was chosen.

num_grat_arp, num_unsol_na

Specify the number of peer notifications (gratuitous ARPs and unsolicited IPv6 Neighbor Advertisements) to be issued after a failover event. As soon as the link is up on the new slave (possibly immediately) a peer notification is sent on the bonding device and each VLAN sub-device. This is repeated at the rate specified by peer_notif_delay if the number is greater than 1.

The valid range is 0 - 255; the default value is 1. These options affect only the active-backup mode. These options were added for bonding versions 3.3.0 and 3.4.0 respectively.

From Linux 3.0 and bonding version 3.7.1, these notifications are generated by the ipv4 and ipv6 code and the numbers of repetitions cannot be set independently.

packets_per_slave

Specify the number of packets to transmit through a slave before moving to the next one. When set to 0 then a slave is chosen at random.

The valid range is 0 - 65535; the default value is 1. This option has effect only in balance-rr mode.

peer_notif_delay

Specify the delay, in milliseconds, between each peer notification (gratuitous ARP and unsolicited IPv6 Neighbor Advertisement) when they are issued after a failover event. This delay should be a multiple of the link monitor interval (arp_interval or miimon, whichever is active). The default value is 0 which means to match the value of the link monitor interval.

primary

A string (eth0, eth2, etc) 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(1), balance-tlb (5) and balance-alb (6) mode.

primary_reselect

Specifies the reselection policy for the primary slave. This affects how the primary slave is chosen to become the active slave when failure of the active slave or recovery of the primary slave occurs. This option is designed to prevent flip-flopping between the primary slave and other slaves. Possible values are:

always or 0 (default)

The primary slave becomes the active slave whenever it comes back up.

better or 1

The primary slave becomes the active slave when it comes back up, if the speed and duplex of the primary slave is better than the speed and duplex of the current active slave.

failure or 2

The primary slave becomes the active slave only if the current active slave fails and the primary slave is up.

The primary_reselect setting is ignored in two cases:

If no slaves are active, the first slave to recover is made the active slave.

When initially enslaved, the primary slave is always made the active slave.

Changing the primary_reselect policy via sysfs will cause an immediate selection of the best active slave according to the new policy. This may or may not result in a change of the active slave, depending upon the circumstances.

This option was added for bonding version 3.6.0.

tlb_dynamic_lb

Specifies if dynamic shuffling of flows is enabled in tlb mode. The value has no effect on any other modes.

The default behavior of tlb mode is to shuffle active flows across slaves based on the load in that interval. This gives nice lb characteristics but can cause packet reordering. If re-ordering is a concern use this variable to disable flow shuffling and rely on load balancing provided solely by the hash distribution. xmit-hash-policy can be used to select the appropriate hashing for the setup.

The sysfs entry can be used to change the setting per bond device and the initial value is derived from the module parameter. The sysfs entry is allowed to be changed only if the bond device is down.

The default value is "1" that enables flow shuffling while value "0" disables it. This option was added in bonding driver 3.7.1

updelay

Specifies the time, in milliseconds, to wait before enabling a slave after a link recovery has been detected. This option is only valid for the miimon link monitor. The updelay value should be a multiple of the miimon value; if not, it will be rounded down to the nearest multiple. The default value is 0.

use_carrier

Specifies whether or not miimon should use MII or ETHTOOL ioctls vs. netif_carrier_ok() to determine the link status. The MII or ETHTOOL ioctls are less efficient and utilize a deprecated calling sequence within the kernel. The netif_carrier_ok() relies on the device driver to maintain its state with netif_carrier_on/off; at this writing, most, but not all, device drivers support this facility.

If bonding insists that the link is up when it should not be, it may be that your network device driver does not support netif_carrier_on/off. The default state for netif_carrier is "carrier on," so if a driver does not support netif_carrier, it will appear as if the link is always up. In this case, setting use_carrier to 0 will cause bonding to revert to the MII / ETHTOOL ioctl method to determine the link state.

A value of 1 enables the use of netif_carrier_ok(), a value of 0 will use the deprecated MII / ETHTOOL ioctls. The default value is 1.

xmit_hash_policy

Selects the transmit hash policy to use for slave selection in balance-xor, 802.3ad, and tlb modes. Possible values are:

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.

encap2+3

This policy uses the same formula as layer2+3 but it relies on skb_flow_dissect to obtain the header fields which might result in the use of inner headers if an encapsulation protocol is used. For example this will improve the performance for tunnel users because the packets will be distributed according to the encapsulated flows.

encap3+4

This policy uses the same formula as layer3+4 but it relies on skb_flow_dissect to obtain the header fields which might result in the use of inner headers if an encapsulation protocol is used. For example this will improve the performance for tunnel users because the packets will be distributed according to the encapsulated flows.

vlan+srcmac

This policy uses a very rudimentary vlan ID and source mac hash to load-balance traffic per-vlan, with failover should one leg fail. The intended use case is for a bond shared by multiple virtual machines, all configured to use their own vlan, to give lacp-like functionality without requiring lacp-capable switching hardware.

The formula for the hash is simply

hash = (vlan ID) XOR (source MAC vendor) XOR (source MAC dev)

The default value is layer2. This option was added in bonding version 2.6.3. In earlier versions of bonding, this parameter does not exist, and the layer2 policy is the only policy. The layer2+3 value was added for bonding version 3.2.2.

resend_igmp

Specifies the number of IGMP membership reports to be issued after a failover event. One membership report is issued immediately after the failover, subsequent packets are sent in each 200ms interval.

The valid range is 0 - 255; the default value is 1. A value of 0 prevents the IGMP membership report from being issued in response to the failover event.

This option is useful for bonding modes balance-rr (0), active-backup (1), balance-tlb (5) and balance-alb (6), in which a failover can switch the IGMP traffic from one slave to another. Therefore a fresh IGMP report must be issued to cause the switch to forward the incoming IGMP traffic over the newly selected slave.

This option was added for bonding version 3.7.0.

lp_interval

Specifies the number of seconds between instances where the bonding driver sends learning packets to each slaves peer switch.

The valid range is 1 - 0x7fffffff; the default value is 1. This Option has effect only in balance-tlb and balance-alb modes.

3. Configuring Bonding Devices

You can configure bonding using either your distro's network initialization scripts, or manually using either iproute2 or the sysfs interface. Distros generally use one of three packages for the network initialization scripts: initscripts, sysconfig or interfaces. Recent versions of these packages have support for bonding, while older versions do not.

We will first describe the options for configuring bonding for distros using versions of initscripts, sysconfig and interfaces with full or partial support for bonding, then provide information on enabling bonding without support from the network initialization scripts (i.e., older versions of initscripts or sysconfig).

If you're unsure whether your distro uses sysconfig, initscripts or interfaces, or don't know if it's new enough, have no fear. Determining this is fairly straightforward.

First, look for a file called interfaces in /etc/network directory. If this file is present in your system, then your system use interfaces. See Configuration with Interfaces Support.

Else, issue the command:

```
$ rpm -qf /sbin/ifup
```

It will respond with a line of text starting with either "initscripts" or "sysconfig," followed by some numbers. This is the package that provides your network initialization scripts.

Next, to determine if your installation supports bonding, issue the command:

```
$ grep ifenslave /sbin/ifup
```

If this returns any matches, then your initscripts or sysconfig has support for bonding.

3.1 Configuration with Sysconfig Support

This section applies to distros using a version of sysconfig with bonding support, for example, SuSE Linux Enterprise Server 9.

SuSE SLES 9's networking configuration system does support bonding, however, at this writing, the YaST system configuration front end does not provide any means to work with bonding devices. Bonding devices can be managed by hand, however, as follows.

First, if they have not already been configured, configure the slave devices. On SLES 9, this is most easily done by running the yast2 sysconfig configuration utility. The goal is for to create an ifcfg-id file for each slave device. The simplest way to accomplish this is to configure the devices for DHCP (this is only to get the file ifcfg-id file created; see below for some issues with DHCP). The name of the configuration file for each device will be of the form:

```
ifcfg-id-xx:xx:xx:xx:xx:xx
```

Where the "xx" portion will be replaced with the digits from the device's permanent MAC address.

Once the set of ifcfg-id-xx:xx:xx:xx:xx:xx files has been created, it is necessary to edit the configuration files for the slave devices (the MAC addresses correspond to those of the slave devices). Before editing, the file will contain multiple lines, and will look something like this:

```
BOOTPROTO='dhcp'
STARTMODE='on'
USERCTL='no'
UNIQUE='XNzu.WeZGOGF+4wE'
_nm_name='bus-pci-0001:61:01.0'
```

Change the BOOTPROTO and STARTMODE lines to the following:

```
BOOTPROTO='none'
STARTMODE='off'
```

Do not alter the UNIQUE or _nm_name lines. Remove any other lines (USERCTL, etc).

Once the ifcfg-id-xx:xx:xx:xx:xx:xx files have been modified, it's time to create the configuration file for the bonding device itself. This file is named ifcfg-bondX, where X is the number of the bonding device to create, starting at 0. The first such file is ifcfg-bond0, the second is ifcfg-bond1, and so on. The sysconfig network configuration system will correctly start multiple instances of bonding.

The contents of the ifcfg-bondX file is as follows:

```
BOOTPROTO="static"
BROADCAST="10.0.2.255"
```

```
    IPADDR="10.0.2.10"
    NETMASK="255.255.0.0"
    NETWORK="10.0.2.0"
    REMOTE_IPADDR=""
    STARTMODE="onboot"
    BONDING_MASTER="yes"
    BONDING_MODULE_OPTS="mode=active-backup miimon=100"
    BONDING_SLAVE0="eth0"
    BONDING_SLAVE1="bus-pci-0000:06:08.1"
```

Replace the sample BROADCAST, IPADDR, NETMASK and NETWORK values with the appropriate values for your network.

The STARTMODE specifies when the device is brought online. The possible values are:

onboot	The device is started at boot time. If you're not sure, this is probably what you want.
manual	The device is started only when ifup is called manually. Bonding devices may be configured this way if you do not wish them to start automatically at boot for some reason.
hotplug	The device is started by a hotplug event. This is not a valid choice for a bonding device.
off or	The device configuration is ignored.
ignore	

The line BONDING_MASTER='yes' indicates that the device is a bonding master device. The only useful value is "yes."

The contents of BONDING_MODULE_OPTS are supplied to the instance of the bonding module for this device. Specify the options for the bonding mode, link monitoring, and so on here. Do not include the max_bonds bonding parameter; this will confuse the configuration system if you have multiple bonding devices.

Finally, supply one BONDING_SLAVEn="slave device" for each slave. where "n" is an increasing value, one for each slave. The "slave device" is either an interface name, e.g., "eth0", or a device specifier for the network device. The interface name is easier to find, but the ethN names are subject to change at boot time if, e.g., a device early in the sequence has failed. The device specifiers (bus-pci-0000:06:08.1 in the example above) specify the physical network device, and will not change unless the device's bus location changes (for example, it is moved from one PCI slot to another). The example above uses one of each type for demonstration purposes; most configurations will choose one or the other for all slave devices.

When all configuration files have been modified or created, networking must be restarted for the configuration changes to take effect. This can be accomplished via the following:

```
    # /etc/init.d/network restart
```

Note that the network control script (/sbin/ifdown) will remove the bonding module as part of the network shutdown processing, so it is not necessary to remove the module by hand if, e.g., the module parameters have changed.

Also, at this writing, YaST/YaST2 will not manage bonding devices (they do not show bonding interfaces on its list of network devices). It is necessary to edit the configuration file by hand to change the bonding configuration.

Additional general options and details of the ifcfg file format can be found in an example ifcfg template file:

```
    /etc/sysconfig/network/ifcfg.template
```

Note that the template does not document the various `BONDING_*` settings described above, but does describe many of the other options.

3.1.1 Using DHCP with Sysconfig

Under sysconfig, configuring a device with BOOTPROTO='dhcp' will cause it to query DHCP for its IP address information. At this writing, this does not function for bonding devices; the scripts attempt to obtain the device address from DHCP prior to adding any of the slave devices. Without active slaves, the DHCP requests are not sent to the network.

3.1.2 Configuring Multiple Bonds with Sysconfig

The sysconfig network initialization system is capable of handling multiple bonding devices. All that is necessary is for each bonding instance to have an appropriately configured ifcfg-bondX file (as described above). Do not specify the "max_bonds" parameter to any instance of bonding, as this will confuse sysconfig. If you require multiple bonding devices with identical parameters, create multiple ifcfg-bondX files.

Because the sysconfig scripts supply the bonding module options in the ifcfg-bondX file, it is not necessary to add them to the system `/etc/modules.d/*.conf` configuration files.

3.2 Configuration with Initscripts Support

This section applies to distros using a recent version of initscripts with bonding support, for example, Red Hat Enterprise Linux version 3 or later, Fedora, etc. On these systems, the network initialization scripts have knowledge of bonding, and can be configured to control bonding devices. Note that older versions of the initscripts package have lower levels of support for bonding; this will be noted where applicable.

These distros will not automatically load the network adapter driver unless the ethX device is configured with an IP address. Because of this constraint, users must manually configure a network-script file for all physical adapters that will be members of a bondX link. Network script files are located in the directory:

/etc/sysconfig/network-scripts

The file name must be prefixed with "ifcfg-eth" and suffixed with the adapter's physical adapter number. For example, the script for eth0 would be named /etc/sysconfig/network-scripts/ifcfg-eth0. Place the following text in the file:

```
DEVICE=eth0
USERCTL=no
ONBOOT=yes
MASTER=bond0
SLAVE=yes
BOOTPROTO=none
```

The DEVICE= line will be different for every ethX device and must correspond with the name of the file, i.e., ifcfg-eth1 must have a device line of DEVICE=eth1. The setting of the MASTER= line will also depend on the final bonding interface name chosen for your bond. As with other network devices, these typically start at 0, and go up one for each device, i.e., the first bonding instance is bond0, the second is bond1, and so on.

Next, create a bond network script. The file name for this script will be /etc/sysconfig/network-scripts/ifcfg-bondX where X is the number of the bond. For bond0 the file is named "ifcfg-bond0", for bond1 it is named "ifcfg-bond1", and so on. Within that file, place the following text:

```
DEVICE=bond0
IPADDR=192.168.1.1
NETMASK=255.255.255.0
NETWORK=192.168.1.0
BROADCAST=192.168.1.255
ONBOOT=yes
BOOTPROTO=none
USERCTL=no
```

Be sure to change the networking specific lines (IPADDR, NETMASK, NETWORK and BROADCAST) to match your network configuration.

For later versions of initscripts, such as that found with Fedora 7 (or later) and Red Hat Enterprise Linux version 5 (or later), it is possible, and, indeed, preferable, to specify the bonding options in the ifcfg-bond0 file, e.g. a line of the format:

```
BONDING_OPTS="mode=active-backup arp_interval=60 arp_ip_target=192.168.1.254"
```

will configure the bond with the specified options. The options specified in BONDING_OPTS are identical to the bonding module parameters except for the arp_ip_target field when using versions of initscripts older than and 8.57 (Fedora 8) and 8.45.19 (Red Hat Enterprise Linux 5.2). When using older versions each target should be included as a separate option and should be preceded by a '+' to indicate it should be added to the list of queried targets, e.g.,:

```
arp_ip_target=+192.168.1.1 arp_ip_target=+192.168.1.2
```

is the proper syntax to specify multiple targets. When specifying options via BONDING_OPTS, it is not necessary to edit `/etc/modprobe.d/*.conf`.

For even older versions of initscripts that do not support BONDING_OPTS, it is necessary to edit /etc/modprobe.d/*.conf, depending upon your distro) to load the bonding module with your desired options when the bond0 interface is brought up. The following lines in /etc/modprobe.d/*.conf will load the bonding module, and select its options:

alias bond0 bonding options bond0 mode=balance-alb miimon=100

Replace the sample parameters with the appropriate set of options for your configuration.

Finally run "/etc/rc.d/init.d/network restart" as root. This will restart the networking subsystem and your bond link should be now up and running.

3.2.1 Using DHCP with Initscripts

Recent versions of initscripts (the versions supplied with Fedora Core 3 and Red Hat Enterprise Linux 4, or later versions, are reported to work) have support for assigning IP information to bonding devices via DHCP.

To configure bonding for DHCP, configure it as described above, except replace the line "BOOTPROTO=none" with "BOOTPROTO=dhcp" and add a line consisting of "TYPE=Bonding". Note that the TYPE value is case sensitive.

3.2.2 Configuring Multiple Bonds with Initscripts

Initscripts packages that are included with Fedora 7 and Red Hat Enterprise Linux 5 support multiple bonding interfaces by simply specifying the appropriate BONDING_OPTS= in ifcfg-bondX where X is the number of the bond. This support requires sysfs support in the kernel, and a bonding driver of version 3.0.0 or later. Other configurations may not support this method for specifying multiple bonding interfaces; for those instances, see the "Configuring Multiple Bonds Manually" section, below.

3.3 Configuring Bonding Manually with iproute2

This section applies to distros whose network initialization scripts (the sysconfig or initscripts package) do not have specific knowledge of bonding. One such distro is SuSE Linux Enterprise Server version 8.

The general method for these systems is to place the bonding module parameters into a config file in /etc/modprobe.d/ (as appropriate for the installed distro), then add modprobe and/or *ip link* commands to the system's global init script. The name of the global init script differs; for sysconfig, it is /etc/init.d/boot.local and for initscripts it is /etc/rc.d/rc.local.

For example, if you wanted to make a simple bond of two e100 devices (presumed to be eth0 and eth1), and have it persist across reboots, edit the appropriate file (/etc/init.d/boot.local or /etc/rc.d/rc.local), and add the following:

```
modprobe bonding mode=balance-alb miimon=100
modprobe e100
ifconfig bond0 192.168.1.1 netmask 255.255.255.0 up
ip link set eth0 master bond0
ip link set eth1 master bond0
```

Replace the example bonding module parameters and bond0 network configuration (IP address, netmask, etc) with the appropriate values for your configuration.

Unfortunately, this method will not provide support for the ifup and ifdown scripts on the bond devices. To reload the bonding configuration, it is necessary to run the initialization script, e.g.,:

```
# /etc/init.d/boot.local
```

or:

```
# /etc/rc.d/rc.local
```

It may be desirable in such a case to create a separate script which only initializes the bonding configuration, then call that separate script from within boot.local. This allows for bonding to be enabled without re-running the entire global init script.

To shut down the bonding devices, it is necessary to first mark the bonding device itself as being down, then remove the appropriate device driver modules. For our example above, you can do the following:

```
# ifconfig bond0 down
# rmmod bonding
# rmmod e100
```

Again, for convenience, it may be desirable to create a script with these commands.

3.3.1 Configuring Multiple Bonds Manually

This section contains information on configuring multiple bonding devices with differing options for those systems whose network initialization scripts lack support for configuring multiple bonds.

If you require multiple bonding devices, but all with the same options, you may wish to use the "max_bonds" module parameter, documented above.

To create multiple bonding devices with differing options, it is preferable to use bonding parameters exported by sysfs, documented in the section below.

For versions of bonding without sysfs support, the only means to provide multiple instances of bonding with differing options is to load the bonding driver multiple times. Note that current versions of the sysconfig network initialization scripts handle this automatically; if your distro uses these scripts, no special action is needed. See the section Configuring Bonding Devices, above, if you're not sure about your network initialization scripts.

To load multiple instances of the module, it is necessary to specify a different name for each instance (the module loading system requires that every loaded module, even multiple instances of the same module, have a unique name). This is accomplished by supplying multiple sets of bonding options in `/etc/modprobe.d/*.conf`, for example:

```
alias bond0 bonding
options bond0 -o bond0 mode=balance-rr miimon=100

alias bond1 bonding
options bond1 -o bond1 mode=balance-alb miimon=50
```

will load the bonding module two times. The first instance is named "bond0" and creates the bond0 device in balance-rr mode with an miimon of 100. The second instance is named "bond1" and creates the bond1 device in balance-alb mode with an miimon of 50.

In some circumstances (typically with older distributions), the above does not work, and the second bonding instance never sees its options. In that case, the second options line can be substituted as follows:

```
install bond1 /sbin/modprobe --ignore-install bonding -o bond1 \
                             mode=balance-alb miimon=50
```

This may be repeated any number of times, specifying a new and unique name in place of bond1 for each subsequent instance.

It has been observed that some Red Hat supplied kernels are unable to rename modules at load time (the "-o bond1" part). Attempts to pass that option to modprobe will produce an "Operation not permitted" error. This has been reported on some Fedora Core kernels, and has been seen on RHEL 4 as well. On kernels exhibiting this problem, it will be impossible to configure multiple bonds with differing parameters (as they are older kernels, and also lack sysfs support).

3.4 Configuring Bonding Manually via Sysfs

Starting with version 3.0.0, Channel Bonding may be configured via the sysfs interface. This interface allows dynamic configuration of all bonds in the system without unloading the module. It also allows for adding and removing bonds at runtime. Ifenslave is no longer required, though it is still supported.

Use of the sysfs interface allows you to use multiple bonds with different configurations without having to reload the module. It also allows you to use multiple, differently configured bonds when bonding is compiled into the kernel.

You must have the sysfs filesystem mounted to configure bonding this way. The examples in this document assume that you are using the standard mount point for sysfs, e.g. /sys. If your sysfs filesystem is mounted elsewhere, you will need to adjust the example paths accordingly.

Creating and Destroying Bonds

To add a new bond foo:

```
# echo +foo > /sys/class/net/bonding_masters
```

To remove an existing bond bar:

```
# echo -bar > /sys/class/net/bonding_masters
```

To show all existing bonds:

```
# cat /sys/class/net/bonding_masters
```

Note

due to 4K size limitation of sysfs files, this list may be truncated if you have more than a few hundred bonds. This is unlikely to occur under normal operating conditions.

Adding and Removing Slaves

Interfaces may be enslaved to a bond using the file /sys/class/net/<bond>/bonding/slaves. The semantics for this file are the same as for the bonding_masters file.

To enslave interface eth0 to bond bond0:

```
# ifconfig bond0 up
# echo +eth0 > /sys/class/net/bond0/bonding/slaves
```

To free slave eth0 from bond bond0:

```
# echo -eth0 > /sys/class/net/bond0/bonding/slaves
```

When an interface is enslaved to a bond, symlinks between the two are created in the sysfs filesystem. In this case, you would get /sys/class/net/bond0/slave_eth0 pointing to /sys/class/net/eth0, and /sys/class/net/eth0/master pointing to /sys/class/net/bond0.

This means that you can tell quickly whether or not an interface is enslaved by looking for the master symlink. Thus: # echo -eth0 > /sys/class/net/eth0/master/bonding/slaves will free eth0 from whatever bond it is enslaved to, regardless of the name of the bond interface.

Changing a Bond's Configuration

Each bond may be configured individually by manipulating the files located in /sys/class/net/<bond name>/bonding

The names of these files correspond directly with the command- line parameters described elsewhere in this file, and, with the exception of arp_ip_target, they accept the same values. To see the current setting, simply cat the appropriate file.

A few examples will be given here; for specific usage guidelines for each parameter, see the appropriate section in this document.

To configure bond0 for balance-alb mode:

```
# ifconfig bond0 down
# echo 6 > /sys/class/net/bond0/bonding/mode
- or -
# echo balance-alb > /sys/class/net/bond0/bonding/mode
```

Note

The bond interface must be down before the mode can be changed.

To enable MII monitoring on bond0 with a 1 second interval:

```
# echo 1000 > /sys/class/net/bond0/bonding/miimon
```

Note

If ARP monitoring is enabled, it will disabled when MII monitoring is enabled, and vice-versa.

To add ARP targets:

```
# echo +192.168.0.100 > /sys/class/net/bond0/bonding/arp_ip_target
# echo +192.168.0.101 > /sys/class/net/bond0/bonding/arp_ip_target
```

Note

up to 16 target addresses may be specified.

To remove an ARP target:

```
# echo -192.168.0.100 > /sys/class/net/bond0/bonding/arp_ip_target
```

To configure the interval between learning packet transmits:

```
# echo 12 > /sys/class/net/bond0/bonding/lp_interval
```

Note

the lp_interval is the number of seconds between instances where the bonding driver sends learning packets to each slaves peer switch. The default interval is 1 second.

Example Configuration

We begin with the same example that is shown in section 3.3, executed with sysfs, and without using ifenslave.

To make a simple bond of two e100 devices (presumed to be eth0 and eth1), and have it persist across reboots, edit the appropriate file (/etc/init.d/boot.local or /etc/rc.d/rc.local), and add the following:

```
modprobe bonding
modprobe e100
echo balance-alb > /sys/class/net/bond0/bonding/mode
ifconfig bond0 192.168.1.1 netmask 255.255.255.0 up
echo 100 > /sys/class/net/bond0/bonding/miimon
echo +eth0 > /sys/class/net/bond0/bonding/slaves
echo +eth1 > /sys/class/net/bond0/bonding/slaves
```

To add a second bond, with two e1000 interfaces in active-backup mode, using ARP monitoring, add the following lines to your init script:

```
modprobe e1000
echo +bond1 > /sys/class/net/bonding_masters
echo active-backup > /sys/class/net/bond1/bonding/mode
ifconfig bond1 192.168.2.1 netmask 255.255.255.0 up
echo +192.168.2.100 /sys/class/net/bond1/bonding/arp_ip_target
echo 2000 > /sys/class/net/bond1/bonding/arp_interval
echo +eth2 > /sys/class/net/bond1/bonding/slaves
echo +eth3 > /sys/class/net/bond1/bonding/slaves
```

3.5 Configuration with Interfaces Support

This section applies to distros which use /etc/network/interfaces file to describe network interface configuration, most notably Debian and it's derivatives.

The ifup and ifdown commands on Debian don't support bonding out of the box. The ifenslave-2.6 package should be installed to provide bonding support. Once installed, this package will provide `bond-*` options to be used into /etc/network/interfaces.

Note that ifenslave-2.6 package will load the bonding module and use the ifenslave command when appropriate.

Example Configurations

In /etc/network/interfaces, the following stanza will configure bond0, in active-backup mode, with eth0 and eth1 as slaves:

```
auto bond0
iface bond0 inet dhcp
        bond-slaves eth0 eth1
        bond-mode active-backup
        bond-miimon 100
        bond-primary eth0 eth1
```

If the above configuration doesn't work, you might have a system using upstart for system startup. This is most notably true for recent Ubuntu versions. The following stanza in /etc/network/interfaces will produce the same result on those systems:

```
auto bond0
iface bond0 inet dhcp
        bond-slaves none
        bond-mode active-backup
        bond-miimon 100

auto eth0
iface eth0 inet manual
        bond-master bond0
        bond-primary eth0 eth1

auto eth1
iface eth1 inet manual
        bond-master bond0
        bond-primary eth0 eth1
```

For a full list of `bond-*` supported options in /etc/network/interfaces and some more advanced examples tailored to you particular distros, see the files in /usr/share/doc/ifenslave-2.6.

3.6 Overriding Configuration for Special Cases

When using the bonding driver, the physical port which transmits a frame is typically selected by the bonding driver, and is not relevant to the user or system administrator. The output port is simply selected using the policies of the selected bonding mode. On occasion however, it is helpful to direct certain classes of traffic to certain physical interfaces on output to implement slightly more complex policies. For example, to reach a web server over a bonded interface in which eth0 connects to a private network, while eth1 connects via a public network, it may be desirous to bias the bond to send said traffic over eth0 first, using eth1 only as a fall back, while all other traffic can safely be sent over either interface. Such configurations may be achieved using the traffic control utilities inherent in linux.

By default the bonding driver is multiqueue aware and 16 queues are created when the driver initializes (see Documentation/networking/multiqueue.rst for details). If more or less queues are desired the module parameter tx_queues can be used to change this value. There is no sysfs parameter available as the allocation is done at module init time.

The output of the file /proc/net/bonding/bondX has changed so the output Queue ID is now printed for each slave:

```
Bonding Mode: fault-tolerance (active-backup)
Primary Slave: None
Currently Active Slave: eth0
MII Status: up
MII Polling Interval (ms): 0
Up Delay (ms): 0
Down Delay (ms): 0

Slave Interface: eth0
MII Status: up
Link Failure Count: 0
Permanent HW addr: 00:1a:a0:12:8f:cb
Slave queue ID: 0

Slave Interface: eth1
MII Status: up
Link Failure Count: 0
Permanent HW addr: 00:1a:a0:12:8f:cc
Slave queue ID: 2
```

The queue_id for a slave can be set using the command:

```
# echo "eth1:2" > /sys/class/net/bond0/bonding/queue_id
```

Any interface that needs a queue_id set should set it with multiple calls like the one above until proper priorities are set for all interfaces. On distributions that allow configuration via initscripts, multiple 'queue_id' arguments can be added to BONDING_OPTS to set all needed slave queues.

These queue id's can be used in conjunction with the tc utility to configure a multiqueue qdisc and filters to bias certain traffic to transmit on certain slave devices. For instance, say we wanted, in the above configuration to force all traffic bound to 192.168.1.100 to use eth1 in the bond as its output device. The following commands would accomplish this:

```
# tc qdisc add dev bond0 handle 1 root multiq

# tc filter add dev bond0 protocol ip parent 1: prio 1 u32 match ip \
        dst 192.168.1.100 action skbedit queue_mapping 2
```

These commands tell the kernel to attach a multiqueue queue discipline to the bond0 interface and filter traffic enqueued to it, such that packets with a dst ip of 192.168.1.100 have their output queue mapping value overwritten to 2. This value is then passed into the driver, causing the normal output path selection policy to be overridden, selecting instead qid 2, which maps to eth1.

Note that qid values begin at 1. Qid 0 is reserved to initiate to the driver that normal output policy selection should take place. One benefit to simply leaving the qid for a slave to 0 is the multiqueue awareness in the bonding driver that is now present. This awareness allows tc filters to be placed on slave devices as well as bond devices and the bonding driver will simply act as a pass-through for selecting output queues on the slave device rather than output port selection.

This feature first appeared in bonding driver version 3.7.0 and support for output slave selection was limited to round-robin and active-backup modes.

3.7 Configuring LACP for 802.3ad mode in a more secure way

When using 802.3ad bonding mode, the Actor (host) and Partner (switch) exchange LACPDUs. These LACPDUs cannot be sniffed, because they are destined to link local mac addresses (which switches/bridges are not supposed to forward). However, most of the values are easily predictable or are simply the machine's MAC address (which is trivially known to all other hosts in the same L2). This implies that other machines in the L2 domain can spoof LACPDU packets from other hosts to the switch and potentially cause mayhem by joining (from the point of view of the switch) another machine's aggregate, thus receiving a portion of that hosts incoming traffic and / or spoofing traffic from that machine themselves (potentially even successfully terminating some portion of flows). Though this is not a likely scenario, one could avoid this possibility by simply configuring few bonding parameters:

a. ad_actor_system : You can set a random mac-address that can be used for these LACPDU exchanges. The value can not be either NULL or Multicast. Also it's preferable to set the local-admin bit. Following shell code generates a random mac-address as described above:

```
# sys_mac_addr=$(printf '%02x:%02x:%02x:%02x:%02x:%02x' \
                         $(( (RANDOM & 0xFE) | 0x02 )) \
                         $(( RANDOM & 0xFF )) \
                         $(( RANDOM & 0xFF )) \
                         $(( RANDOM & 0xFF )) \
                         $(( RANDOM & 0xFF )) \
                         $(( RANDOM & 0xFF )))
# echo $sys_mac_addr > /sys/class/net/bond0/bonding/ad_actor_system
```

b. ad_actor_sys_prio : Randomize the system priority. The default value is 65535, but system can take the value from 1 - 65535. Following shell code generates random priority and sets it:

```
# sys_prio=$(( 1 + RANDOM + RANDOM ))
# echo $sys_prio > /sys/class/net/bond0/bonding/ad_actor_sys_prio
```

c. ad_user_port_key : Use the user portion of the port-key. The default keeps this empty. These are the upper 10 bits of the port-key and value ranges from 0 - 1023. Following shell code generates these 10 bits and sets it:

```
# usr_port_key=$(( RANDOM & 0x3FF ))
# echo $usr_port_key > /sys/class/net/bond0/bonding/ad_user_port_key
```

4 Querying Bonding Configuration

4.1 Bonding Configuration

Each bonding device has a read-only file residing in the /proc/net/bonding directory. The file contents include information about the bonding configuration, options and state of each slave.

For example, the contents of /proc/net/bonding/bond0 after the driver is loaded with parameters of mode=0 and miimon=1000 is generally as follows:

```
Ethernet Channel Bonding Driver: 2.6.1 (October 29, 2004)
Bonding Mode: load balancing (round-robin)
Currently Active Slave: eth0
MII Status: up
MII Polling Interval (ms): 1000
Up Delay (ms): 0
Down Delay (ms): 0

Slave Interface: eth1
MII Status: up
Link Failure Count: 1

Slave Interface: eth0
MII Status: up
Link Failure Count: 1
```

The precise format and contents will change depending upon the bonding configuration, state, and version of the bonding driver.

4.2 Network configuration

4.2 Network configuration

The network configuration can be inspected using the ifconfig command. Bonding devices will have the MASTER flag set; Bonding slave devices will have the SLAVE flag set. The ifconfig output does not contain information on which slaves are associated with which masters.

In the example below, the bond0 interface is the master (MASTER) while eth0 and eth1 are slaves (SLAVE). Notice all slaves of bond0 have the same MAC address (HWaddr) as bond0 for all modes except TLB and ALB that require a unique MAC address for each slave:

```
# /sbin/ifconfig
bond0     Link encap:Ethernet  HWaddr 00:C0:F0:1F:37:B4
          inet addr:XXX.XXX.XXX.YYY  Bcast:XXX.XXX.XXX.255  Mask:255.255.252.0
          UP BROADCAST RUNNING MASTER MULTICAST  MTU:1500  Metric:1
          RX packets:7224794 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3286647 errors:1 dropped:0 overruns:1 carrier:0
          collisions:0 txqueuelen:0

eth0      Link encap:Ethernet  HWaddr 00:C0:F0:1F:37:B4
          UP BROADCAST RUNNING SLAVE MULTICAST  MTU:1500  Metric:1
          RX packets:3573025 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1643167 errors:1 dropped:0 overruns:1 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:10 Base address:0x1080

eth1      Link encap:Ethernet  HWaddr 00:C0:F0:1F:37:B4
          UP BROADCAST RUNNING SLAVE MULTICAST  MTU:1500  Metric:1
          RX packets:3651769 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1643480 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:9 Base address:0x1400
```

5. Switch Configuration

For this section, "switch" refers to whatever system the bonded devices are directly connected to (i.e., where the other end of the cable plugs into). This may be an actual dedicated switch device, or it may be another regular system (e.g., another computer running Linux),

The active-backup, balance-tlb and balance-alb modes do not require any specific configuration of the switch.

The 802.3ad mode requires that the switch have the appropriate ports configured as an 802.3ad aggregation. The precise method used to configure this varies from switch to switch, but, for example, a Cisco 3550 series switch requires that the appropriate ports first be grouped together in a single etherchannel instance, then that etherchannel is set to mode "lacp" to enable 802.3ad (instead of standard EtherChannel).

The balance-rr, balance-xor and broadcast modes generally require that the switch have the appropriate ports grouped together. The nomenclature for such a group differs between switches, it may be called an "etherchannel" (as in the Cisco example, above), a "trunk group" or some other similar variation. For these modes, each switch will also have its own configuration options for the switch's transmit policy to the bond. Typical choices include XOR of either the MAC or IP addresses. The transmit policy of the two peers does not need to match. For these three modes, the bonding mode really selects a transmit policy for an EtherChannel group; all three will interoperate with another EtherChannel group.

6. 802.1q VLAN Support

It is possible to configure VLAN devices over a bond interface using the 8021q driver. However, only packets coming from the 8021q driver and passing through bonding will be tagged by default. Self generated packets, for example, bonding's learning packets or ARP packets generated by either ALB mode or the ARP monitor mechanism, are tagged internally by bonding itself. As a result, bonding must "learn" the VLAN IDs configured above it, and use those IDs to tag self generated packets.

For reasons of simplicity, and to support the use of adapters that can do VLAN hardware acceleration offloading, the bonding interface declares itself as fully hardware offloading capable, it gets the add_vid/kill_vid notifications to gather the necessary information, and it propagates those actions to the slaves. In case of mixed adapter types, hardware accelerated tagged packets that should go through an adapter that is not offloading capable are "un-accelerated" by the bonding driver so the VLAN tag sits in the regular location.

VLAN interfaces *must* be added on top of a bonding interface only after enslaving at least one slave. The bonding interface has a hardware address of 00:00:00:00:00:00 until the first slave is added. If the VLAN interface is created prior to the first enslavement, it would pick up the all-zeroes hardware address. Once the first slave is attached to the bond, the bond device itself will pick up the slave's hardware address, which is then available for the VLAN device.

Also, be aware that a similar problem can occur if all slaves are released from a bond that still has one or more VLAN interfaces on top of it. When a new slave is added, the bonding interface will obtain its hardware address from the first slave, which might not match the hardware address of the VLAN interfaces (which was ultimately copied from an earlier slave).

There are two methods to insure that the VLAN device operates with the correct hardware address if all slaves are removed from a bond interface:

1. Remove all VLAN interfaces then recreate them

2. Set the bonding interface's hardware address so that it matches the hardware address of the VLAN interfaces.

Note that changing a VLAN interface's HW address would set the underlying device -- i.e. the bonding interface -- to promiscuous mode, which might not be what you want.

7. Link Monitoring

The bonding driver at present supports two schemes for monitoring a slave device's link state: the ARP monitor and the MII monitor.

At the present time, due to implementation restrictions in the bonding driver itself, it is not possible to enable both ARP and MII monitoring simultaneously.

7.1 ARP Monitor Operation

The ARP monitor operates as its name suggests: it sends ARP queries to one or more designated peer systems on the network, and uses the response as an indication that the link is operating. This gives some assurance that traffic is actually flowing to and from one or more peers on the local network.

The ARP monitor relies on the device driver itself to verify that traffic is flowing. In particular, the driver must keep up to date the last receive time, dev->last_rx. Drivers that use NETIF_F_LLTX flag must also update netdev_queue->trans_start. If they do not, then the ARP monitor will immediately fail any slaves using that driver, and those slaves will stay down. If networking monitoring (tcpdump, etc) shows the ARP requests and replies on the network, then it may be that your device driver is not updating last_rx and trans_start.

7.2 Configuring Multiple ARP Targets

While ARP monitoring can be done with just one target, it can be useful in a High Availability setup to have several targets to monitor. In the case of just one target, the target itself may go down or have a problem making it unresponsive to ARP requests. Having an additional target (or several) increases the reliability of the ARP monitoring.

Multiple ARP targets must be separated by commas as follows:

```
# example options for ARP monitoring with three targets
alias bond0 bonding
options bond0 arp_interval=60 arp_ip_target=192.168.0.1,192.168.0.3,192.168.0.9
```

For just a single target the options would resemble:

```
# example options for ARP monitoring with one target
alias bond0 bonding
options bond0 arp_interval=60 arp_ip_target=192.168.0.100
```

7.3 MII Monitor Operation

The MII monitor monitors only the carrier state of the local network interface. It accomplishes this in one of three ways: by depending upon the device driver to maintain its carrier state, by querying the device's MII registers, or by making an ethtool query to the device.

If the use_carrier module parameter is 1 (the default value), then the MII monitor will rely on the driver for carrier state information (via the netif_carrier subsystem). As explained in the use_carrier parameter information, above, if the MII monitor fails to detect carrier loss on the device (e.g., when the cable is physically disconnected), it may be that the driver does not support netif_carrier.

If use_carrier is 0, then the MII monitor will first query the device's (via ioctl) MII registers and check the link state. If that request fails (not just that it returns carrier down), then the MII monitor will make an ethtool ETHTOOL_GLINK request to attempt to obtain the same information. If both methods fail (i.e., the driver either does not support or had some error in processing both the MII register and ethtool requests), then the MII monitor will assume the link is up.

8. Potential Sources of Trouble

8.1 Adventures in Routing

When bonding is configured, it is important that the slave devices not have routes that supersede routes of the master (or, generally, not have routes at all). For example, suppose the bonding device bond0 has two slaves, eth0 and eth1, and the routing table is as follows:

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.0.0        0.0.0.0         255.255.0.0     U        40 0          0 eth0
10.0.0.0        0.0.0.0         255.255.0.0     U        40 0          0 eth1
10.0.0.0        0.0.0.0         255.255.0.0     U        40 0          0 bond0
127.0.0.0       0.0.0.0         255.0.0.0       U        40 0          0 lo
```

This routing configuration will likely still update the receive/transmit times in the driver (needed by the ARP monitor), but may bypass the bonding driver (because outgoing traffic to, in this case, another host on network 10 would use eth0 or eth1 before bond0).

The ARP monitor (and ARP itself) may become confused by this configuration, because ARP requests (generated by the ARP monitor) will be sent on one interface (bond0), but the corresponding reply will arrive on a different interface (eth0). This reply looks to ARP as an unsolicited ARP reply (because ARP matches replies on an interface basis), and is discarded. The MII monitor is not affected by the state of the routing table.

The solution here is simply to insure that slaves do not have routes of their own, and if for some reason they must, those routes do not supersede routes of their master. This should generally be the case, but unusual configurations or errant manual or automatic static route additions may cause trouble.

8.2 Ethernet Device Renaming

On systems with network configuration scripts that do not associate physical devices directly with network interface names (so that the same physical device always has the same "ethX" name), it may be necessary to add some special logic to config files in /etc/modprobe.d/.

For example, given a modules.conf containing the following:

```
alias bond0 bonding
options bond0 mode=some-mode miimon=50
alias eth0 tg3
alias eth1 tg3
alias eth2 e1000
alias eth3 e1000
```

If neither eth0 and eth1 are slaves to bond0, then when the bond0 interface comes up, the devices may end up reordered. This happens because bonding is loaded first, then its slave device's drivers are loaded next. Since no other drivers have been loaded, when the e1000 driver loads, it will receive eth0 and eth1 for its devices, but the bonding configuration tries to enslave eth2 and eth3 (which may later be assigned to the tg3 devices).

Adding the following:

```
add above bonding e1000 tg3
```

causes modprobe to load e1000 then tg3, in that order, when bonding is loaded. This command is fully documented in the modules.conf manual page.

On systems utilizing modprobe an equivalent problem can occur. In this case, the following can be added to config files in /etc/modprobe.d/ as:

```
softdep bonding pre: tg3 e1000
```

This will load tg3 and e1000 modules before loading the bonding one. Full documentation on this can be found in the modprobe.d and modprobe manual pages.

8.3. Painfully Slow Or No Failed Link Detection By Miimon

By default, bonding enables the use_carrier option, which instructs bonding to trust the driver to maintain carrier state.

As discussed in the options section, above, some drivers do not support the netif_carrier_on/_off link state tracking system. With use_carrier enabled, bonding will always see these links as up, regardless of their actual state.

Additionally, other drivers do support netif_carrier, but do not maintain it in real time, e.g., only polling the link state at some fixed interval. In this case, miimon will detect failures, but only after some long period of time has expired. If it appears that miimon is very slow in detecting link failures, try specifying use_carrier=0 to see if that improves the failure detection time. If it does, then it may be that the driver checks the carrier state at a fixed interval, but does not cache the MII register values (so the use_carrier=0 method of querying the registers directly works). If use_carrier=0 does not improve the failover, then the driver may cache the registers, or the problem may be elsewhere.

Also, remember that miimon only checks for the device's carrier state. It has no way to determine the state of devices on or beyond other ports of a switch, or if a switch is refusing to pass traffic while still maintaining carrier on.

9. SNMP agents

If running SNMP agents, the bonding driver should be loaded before any network drivers participating in a bond. This requirement is due to the interface index (ipAdEntIfIndex) being associated to the first interface found with a given IP address. That is, there is only one ipAdEntIfIndex for each IP address. For example, if eth0 and eth1 are slaves of bond0 and the driver for eth0 is loaded before the bonding driver, the interface for the IP address will be associated with the eth0 interface. This configuration is shown below, the IP address 192.168.1.1 has an interface index of 2 which indexes to eth0 in the ifDescr table (ifDescr.2).

```
interfaces.ifTable.ifEntry.ifDescr.1 = lo
interfaces.ifTable.ifEntry.ifDescr.2 = eth0
interfaces.ifTable.ifEntry.ifDescr.3 = eth1
```

```
    interfaces.ifTable.ifEntry.ifDescr.4 = eth2
    interfaces.ifTable.ifEntry.ifDescr.5 = eth3
    interfaces.ifTable.ifEntry.ifDescr.6 = bond0
    ip.ipAddrTable.ipAddrEntry.ipAdEntIfIndex.10.10.10.10 = 5
    ip.ipAddrTable.ipAddrEntry.ipAdEntIfIndex.192.168.1.1 = 2
    ip.ipAddrTable.ipAddrEntry.ipAdEntIfIndex.10.74.20.94 = 4
    ip.ipAddrTable.ipAddrEntry.ipAdEntIfIndex.127.0.0.1 = 1
```

This problem is avoided by loading the bonding driver before any network drivers participating in a bond. Below is an example of loading the bonding driver first, the IP address 192.168.1.1 is correctly associated with ifDescr.2.

> interfaces.ifTable.ifEntry.ifDescr.1 = lo interfaces.ifTable.ifEntry.ifDescr.2 = bond0 interfaces.ifTable.ifEntry.ifDescr.3 = eth0 interfaces.ifTable.ifEntry.ifDescr.4 = eth1 interfaces.ifTable.ifEntry.ifDescr.5 = eth2 interfaces.ifTable.ifEntry.ifDescr.6 = eth3 ip.ipAddrTable.ipAddrEntry.ipAdEntIfIndex.10.10.10.10 = 6 ip.ipAddrTable.ipAddrEntry.ipAdEntIfIndex.192.168.1.1 = 2 ip.ipAddrTable.ipAddrEntry.ipAdEntIfIndex.10.74.20.94 = 5 ip.ipAddrTable.ipAddrEntry.ipAdEntIfIndex.127.0.0.1 = 1

While some distributions may not report the interface name in ifDescr, the association between the IP address and IfIndex remains and SNMP functions such as Interface_Scan_Next will report that association.

10. Promiscuous mode

When running network monitoring tools, e.g., tcpdump, it is common to enable promiscuous mode on the device, so that all traffic is seen (instead of seeing only traffic destined for the local host). The bonding driver handles promiscuous mode changes to the bonding master device (e.g., bond0), and propagates the setting to the slave devices.

For the balance-rr, balance-xor, broadcast, and 802.3ad modes, the promiscuous mode setting is propagated to all slaves.

For the active-backup, balance-tlb and balance-alb modes, the promiscuous mode setting is propagated only to the active slave.

For balance-tlb mode, the active slave is the slave currently receiving inbound traffic.

For balance-alb mode, the active slave is the slave used as a "primary." This slave is used for mode-specific control traffic, for sending to peers that are unassigned or if the load is unbalanced.

For the active-backup, balance-tlb and balance-alb modes, when the active slave changes (e.g., due to a link failure), the promiscuous setting will be propagated to the new active slave.

11. Configuring Bonding for High Availability

High Availability refers to configurations that provide maximum network availability by having redundant or backup devices, links or switches between the host and the rest of the world. The goal is to provide the maximum availability of network connectivity (i.e., the network always works), even though other configurations could provide higher throughput.

11.1 High Availability in a Single Switch Topology

If two hosts (or a host and a single switch) are directly connected via multiple physical links, then there is no availability penalty to optimizing for maximum bandwidth. In this case, there is only one switch (or peer), so if it fails, there is no alternative access to fail over to. Additionally, the bonding load balance modes support link monitoring of their members, so if individual links fail, the load will be rebalanced across the remaining devices.

See Section 12, "Configuring Bonding for Maximum Throughput" for information on configuring bonding with one peer device.

11.2 High Availability in a Multiple Switch Topology

With multiple switches, the configuration of bonding and the network changes dramatically. In multiple switch topologies, there is a trade off between network availability and usable bandwidth.

Below is a sample network, configured to maximize the availability of the network:

```
          |                                     |
          |port3                           port3|
    +-----+----+                          +-----+----+
    |          |port2       ISL      port2|          |
    | switch A +--------------------------+ switch B |
    |          |                          |          |
    +-----+----+                          +-----++---+
          |port1                           port1|
          |             +-------+               |
          +-------------+ host1 +---------------+
                   eth0 +-------+ eth1
```

In this configuration, there is a link between the two switches (ISL, or inter switch link), and multiple ports connecting to the outside world ("port3" on each switch). There is no technical reason that this could not be extended to a third switch.

11.2.1 HA Bonding Mode Selection for Multiple Switch Topology

In a topology such as the example above, the active-backup and broadcast modes are the only useful bonding modes when optimizing for availability; the other modes require all links to terminate on the same peer for them to behave rationally.

active-backup:
: This is generally the preferred mode, particularly if the switches have an ISL and play together well. If the network configuration is such that one switch is specifically a backup switch (e.g., has lower capacity, higher cost, etc), then the primary option can be used to insure that the preferred link is always used when it is available.

broadcast:
: This mode is really a special purpose mode, and is suitable only for very specific needs. For example, if the two switches are not connected (no ISL), and the networks beyond them are totally independent. In this case, if it is necessary for some specific one-way traffic to reach both independent networks, then the broadcast mode may be suitable.

11.2.2 HA Link Monitoring Selection for Multiple Switch Topology

The choice of link monitoring ultimately depends upon your switch. If the switch can reliably fail ports in response to other failures, then either the MII or ARP monitors should work. For example, in the above example, if the "port3" link fails at the remote end, the MII monitor has no direct means to detect this. The ARP monitor could be configured with a target at the remote end of port3, thus detecting that failure without switch support.

In general, however, in a multiple switch topology, the ARP monitor can provide a higher level of reliability in detecting end to end connectivity failures (which may be caused by the failure of any individual component to pass traffic for any reason). Additionally, the ARP monitor should be configured with multiple targets (at least one for each switch in the network). This will insure that, regardless of which switch is active, the ARP monitor has a suitable target to query.

Note, also, that of late many switches now support a functionality generally referred to as "trunk failover." This is a feature of the switch that causes the link state of a particular switch port to be set down (or up) when the state of another switch port goes down (or up). Its purpose is to propagate link failures from logically "exterior" ports to the logically "interior" ports that bonding is able to monitor via miimon. Availability and configuration for trunk failover varies by switch, but this can be a viable alternative to the ARP monitor when using suitable switches.

12. Configuring Bonding for Maximum Throughput

12.1 Maximizing Throughput in a Single Switch Topology

In a single switch configuration, the best method to maximize throughput depends upon the application and network environment. The various load balancing modes each have strengths and weaknesses in different environments, as detailed below.

For this discussion, we will break down the topologies into two categories. Depending upon the destination of most traffic, we categorize them into either "gatewayed" or "local" configurations.

In a gatewayed configuration, the "switch" is acting primarily as a router, and the majority of traffic passes through this router to other networks. An example would be the following:

```
    +----------+                     +----------+
    |          |eth0            port1|          | to other networks
    | Host A   +---------------------+ router   +------------------->
    |          +---------------------+          | Hosts B and C are out
    |          |eth1            port2|          | here somewhere
    +----------+                     +----------+
```

The router may be a dedicated router device, or another host acting as a gateway. For our discussion, the important point is that the majority of traffic from Host A will pass through the router to some other network before reaching its final destination.

In a gatewayed network configuration, although Host A may communicate with many other systems, all of its traffic will be sent and received via one other peer on the local network, the router.

Note that the case of two systems connected directly via multiple physical links is, for purposes of configuring bonding, the same as a gatewayed configuration. In that case, it happens that all traffic is destined for the "gateway" itself, not some other network beyond the gateway.

In a local configuration, the "switch" is acting primarily as a switch, and the majority of traffic passes through this switch to reach other stations on the same network. An example would be the following:

```
    +----------+            +----------+       +--------+
    |          |eth0   port1|          +-------+ Host B |
    |  Host A  +------------+  switch  |port3  +--------+
    |          +------------+          |                  +--------+
    |          |eth1   port2|          +------------------+ Host C |
    +----------+            +----------+port4             +--------+
```

Again, the switch may be a dedicated switch device, or another host acting as a gateway. For our discussion, the important point is that the majority of traffic from Host A is destined for other hosts on the same local network (Hosts B and C in the above example).

In summary, in a gatewayed configuration, traffic to and from the bonded device will be to the same MAC level peer on the network (the gateway itself, i.e., the router), regardless of its final destination. In a local configuration, traffic flows directly to and from the final destinations, thus, each destination (Host B, Host C) will be addressed directly by their individual MAC addresses.

This distinction between a gatewayed and a local network configuration is important because many of the load balancing modes available use the MAC addresses of the local network source and destination to make load balancing decisions. The behavior of each mode is described below.

12.1.1 MT Bonding Mode Selection for Single Switch Topology

This configuration is the easiest to set up and to understand, although you will have to decide which bonding mode best suits your needs. The trade offs for each mode are detailed below:

balance-rr:

> This mode is the only mode that will permit a single TCP/IP connection to stripe traffic across multiple interfaces. It is therefore the only mode that will allow a single TCP/IP stream to utilize more than one interface's worth of throughput. This comes at a cost, however: the striping generally results in peer systems receiving packets out of order, causing TCP/IP's congestion control system to kick in, often by retransmitting segments.
>
> It is possible to adjust TCP/IP's congestion limits by altering the net.ipv4.tcp_reordering sysctl parameter. The usual default value is 3. But keep in mind TCP stack is able to automatically increase this when it detects reorders.
>
> Note that the fraction of packets that will be delivered out of order is highly variable, and is unlikely to be zero. The level of reordering depends upon a variety of factors, including the networking interfaces, the switch, and the topology of the configuration. Speaking in general terms, higher speed network cards produce more reordering (due to factors such as packet coalescing), and a "many to many" topology will reorder at a higher rate than a "many slow to one fast" configuration.
>
> Many switches do not support any modes that stripe traffic (instead choosing a port based upon IP or MAC level addresses); for those devices, traffic for a particular connection flowing through the switch to a balance-rr bond will not utilize greater than one interface's worth of bandwidth.
>
> If you are utilizing protocols other than TCP/IP, UDP for example, and your application can tolerate out of order delivery, then this mode can allow for single stream datagram performance that scales near linearly as interfaces are added to the bond.
>
> This mode requires the switch to have the appropriate ports configured for "etherchannel" or "trunking."

active-backup:

> There is not much advantage in this network topology to the active-backup mode, as the inactive backup devices are all connected to the same peer as the primary. In this case, a load balancing mode (with link monitoring) will provide the same level of network availability, but with increased available bandwidth. On the plus side, active-backup mode does not require any configuration of the switch, so it may have value if the hardware available does not support any of the load balance modes.

balance-xor:

> This mode will limit traffic such that packets destined for specific peers will always be sent over the same interface. Since the destination is determined by the MAC addresses involved, this mode works best in a "local" network configuration (as described above), with destinations all on the same local network. This mode is likely to be suboptimal if all your traffic is passed through a single router (i.e., a "gatewayed" network configuration, as described above).
>
> As with balance-rr, the switch ports need to be configured for "etherchannel" or "trunking."

broadcast:

> Like active-backup, there is not much advantage to this mode in this type of network topology.

802.3ad:

> This mode can be a good choice for this type of network topology. The 802.3ad mode is an IEEE standard, so all peers that implement 802.3ad should interoperate well. The 802.3ad protocol includes automatic configuration of the aggregates, so minimal manual configuration of the switch is needed (typically only to designate that some set of devices is available for 802.3ad). The 802.3ad standard also mandates that frames be delivered in order (within certain limits), so in general single connections will not see misordering of packets. The 802.3ad mode does have some drawbacks: the standard mandates that all devices in the aggregate operate at the same speed and duplex. Also, as with all bonding load balance modes other than balance-rr, no single connection will be able to utilize more than a single interface's worth of bandwidth.
>
> Additionally, the linux bonding 802.3ad implementation distributes traffic by peer (using an XOR of MAC addresses and packet type ID), so in a "gatewayed" configuration, all outgoing traffic will generally use the same device. Incoming traffic may also end up on a single device, but that is dependent upon the balancing policy of the peer's 802.3ad implementation. In a "local" configuration, traffic will be distributed across the devices in the bond.
>
> Finally, the 802.3ad mode mandates the use of the MII monitor, therefore, the ARP monitor is not available in this mode.

balance-tlb:

The balance-tlb mode balances outgoing traffic by peer. Since the balancing is done according to MAC address, in a "gatewayed" configuration (as described above), this mode will send all traffic across a single device. However, in a "local" network configuration, this mode balances multiple local network peers across devices in a vaguely intelligent manner (not a simple XOR as in balance-xor or 802.3ad mode), so that mathematically unlucky MAC addresses (i.e., ones that XOR to the same value) will not all "bunch up" on a single interface.

Unlike 802.3ad, interfaces may be of differing speeds, and no special switch configuration is required. On the down side, in this mode all incoming traffic arrives over a single interface, this mode requires certain ethtool support in the network device driver of the slave interfaces, and the ARP monitor is not available.

balance-alb:

This mode is everything that balance-tlb is, and more. It has all of the features (and restrictions) of balance-tlb, and will also balance incoming traffic from local network peers (as described in the Bonding Module Options section, above).

The only additional down side to this mode is that the network device driver must support changing the hardware address while the device is open.

12.1.2 MT Link Monitoring for Single Switch Topology

The choice of link monitoring may largely depend upon which mode you choose to use. The more advanced load balancing modes do not support the use of the ARP monitor, and are thus restricted to using the MII monitor (which does not provide as high a level of end to end assurance as the ARP monitor).

12.2 Maximum Throughput in a Multiple Switch Topology

Multiple switches may be utilized to optimize for throughput when they are configured in parallel as part of an isolated network between two or more systems, for example:

```
                +-----------+
                |  Host A   |
                +-+---+---+-+
                  |   |   |
         +--------+   |   +---------+
         |            |             |
  +------+---+  +-----+----+  +-----+----+
  | Switch A |  | Switch B |  | Switch C |
  +------+---+  +-----+----+  +-----+----+
         |            |             |
         +--------+   |   +---------+
                  |   |   |
                +-+---+---+-+
                |  Host B   |
                +-----------+
```

In this configuration, the switches are isolated from one another. One reason to employ a topology such as this is for an isolated network with many hosts (a cluster configured for high performance, for example), using multiple smaller switches can be more cost effective than a single larger switch, e.g., on a network with 24 hosts, three 24 port switches can be significantly less expensive than a single 72 port switch.

If access beyond the network is required, an individual host can be equipped with an additional network device connected to an external network; this host then additionally acts as a gateway.

12.2.1 MT Bonding Mode Selection for Multiple Switch Topology

In actual practice, the bonding mode typically employed in configurations of this type is balance-rr. Historically, in this network configuration, the usual caveats about out of order packet delivery are mitigated by the use of network adapters that do not do any kind of packet coalescing (via the use of NAPI, or because the device itself does not generate interrupts until some number of packets has arrived). When employed in this fashion, the balance-rr mode allows individual connections between two hosts to effectively utilize greater than one interface's bandwidth.

12.2.2 MT Link Monitoring for Multiple Switch Topology

Again, in actual practice, the MII monitor is most often used in this configuration, as performance is given preference over availability. The ARP monitor will function in this topology, but its advantages over the MII monitor are mitigated by the volume of probes needed as the number of systems involved grows (remember that each host in the network is configured with bonding).

13. Switch Behavior Issues

13.1 Link Establishment and Failover Delays

Some switches exhibit undesirable behavior with regard to the timing of link up and down reporting by the switch.

First, when a link comes up, some switches may indicate that the link is up (carrier available), but not pass traffic over the interface for some period of time. This delay is typically due to some type of autonegotiation or routing protocol, but may also occur during switch initialization (e.g., during recovery after a switch failure). If you find this to be a problem, specify an appropriate value to the updelay bonding module option to delay the use of the relevant interface(s).

Second, some switches may "bounce" the link state one or more times while a link is changing state. This occurs most commonly while the switch is initializing. Again, an appropriate updelay value may help.

Note that when a bonding interface has no active links, the driver will immediately reuse the first link that goes up, even if the updelay parameter has been specified (the updelay is ignored in this case). If there are slave interfaces waiting for the updelay timeout to expire, the interface that first went into that state will be immediately reused. This reduces down time of the network if the value of updelay has been overestimated, and since this occurs only in cases with no connectivity, there is no additional penalty for ignoring the updelay.

In addition to the concerns about switch timings, if your switches take a long time to go into backup mode, it may be desirable to not activate a backup interface immediately after a link goes down. Failover may be delayed via the downdelay bonding module option.

13.2 Duplicated Incoming Packets

NOTE: Starting with version 3.0.2, the bonding driver has logic to suppress duplicate packets, which should largely eliminate this problem. The following description is kept for reference.

It is not uncommon to observe a short burst of duplicated traffic when the bonding device is first used, or after it has been idle for some period of time. This is most easily observed by issuing a "ping" to some other host on the network, and noticing that the output from ping flags duplicates (typically one per slave).

For example, on a bond in active-backup mode with five slaves all connected to one switch, the output may appear as follows:

```
# ping -n 10.0.4.2
PING 10.0.4.2 (10.0.4.2) from 10.0.3.10 : 56(84) bytes of data.
64 bytes from 10.0.4.2: icmp_seq=1 ttl=64 time=13.7 ms
64 bytes from 10.0.4.2: icmp_seq=1 ttl=64 time=13.8 ms (DUP!)
64 bytes from 10.0.4.2: icmp_seq=1 ttl=64 time=13.8 ms (DUP!)
64 bytes from 10.0.4.2: icmp_seq=1 ttl=64 time=13.8 ms (DUP!)
64 bytes from 10.0.4.2: icmp_seq=1 ttl=64 time=13.8 ms (DUP!)
64 bytes from 10.0.4.2: icmp_seq=2 ttl=64 time=0.216 ms
64 bytes from 10.0.4.2: icmp_seq=3 ttl=64 time=0.267 ms
64 bytes from 10.0.4.2: icmp_seq=4 ttl=64 time=0.222 ms
```

This is not due to an error in the bonding driver, rather, it is a side effect of how many switches update their MAC forwarding tables. Initially, the switch does not associate the MAC address in the packet with a particular switch port, and so it may send the traffic to all ports until its MAC forwarding table is updated. Since the interfaces attached to the bond may occupy multiple ports on a single switch, when the switch (temporarily) floods the traffic to all ports, the bond device receives multiple copies of the same packet (one per slave device).

The duplicated packet behavior is switch dependent, some switches exhibit this, and some do not. On switches that display this behavior, it can be induced by clearing the MAC forwarding table (on most Cisco switches, the privileged command "clear mac address-table dynamic" will accomplish this).

14. Hardware Specific Considerations

This section contains additional information for configuring bonding on specific hardware platforms, or for interfacing bonding with particular switches or other devices.

14.1 IBM BladeCenter

This applies to the JS20 and similar systems.

On the JS20 blades, the bonding driver supports only balance-rr, active-backup, balance-tlb and balance-alb modes. This is largely due to the network topology inside the BladeCenter, detailed below.

JS20 network adapter information

All JS20s come with two Broadcom Gigabit Ethernet ports integrated on the planar (that's "motherboard" in IBM-speak). In the BladeCenter chassis, the eth0 port of all JS20 blades is hard wired to I/O Module #1; similarly, all eth1 ports are wired to I/O Module #2. An add-on Broadcom daughter card can be installed on a JS20 to provide two more Gigabit Ethernet ports. These ports, eth2 and eth3, are wired to I/O Modules 3 and 4, respectively.

Each I/O Module may contain either a switch or a passthrough module (which allows ports to be directly connected to an external switch). Some bonding modes require a specific BladeCenter internal network topology in order to function; these are detailed below.

Additional BladeCenter-specific networking information can be found in two IBM Redbooks (www.ibm.com/redbooks):

- "IBM eServer BladeCenter Networking Options"
- "IBM eServer BladeCenter Layer 2-7 Network Switching"

BladeCenter networking configuration

Because a BladeCenter can be configured in a very large number of ways, this discussion will be confined to describing basic configurations.

Normally, Ethernet Switch Modules (ESMs) are used in I/O modules 1 and 2. In this configuration, the eth0 and eth1 ports of a JS20 will be connected to different internal switches (in the respective I/O modules).

A passthrough module (OPM or CPM, optical or copper, passthrough module) connects the I/O module directly to an external switch. By using PMs in I/O module #1 and #2, the eth0 and eth1 interfaces of a JS20 can be redirected to the outside world and connected to a common external switch.

Depending upon the mix of ESMs and PMs, the network will appear to bonding as either a single switch topology (all PMs) or as a multiple switch topology (one or more ESMs, zero or more PMs). It is also possible to connect ESMs together, resulting in a configuration much like the example in "High Availability in a Multiple Switch Topology," above.

Requirements for specific modes

The balance-rr mode requires the use of passthrough modules for devices in the bond, all connected to an common external switch. That switch must be configured for "etherchannel" or "trunking" on the appropriate ports, as is usual for balance-rr.

The balance-alb and balance-tlb modes will function with either switch modules or passthrough modules (or a mix). The only specific requirement for these modes is that all network interfaces must be able to reach all destinations for traffic sent over the bonding device (i.e., the network must converge at some point outside the BladeCenter).

The active-backup mode has no additional requirements.

Link monitoring issues

When an Ethernet Switch Module is in place, only the ARP monitor will reliably detect link loss to an external switch. This is nothing unusual, but examination of the BladeCenter cabinet would suggest that the "external" network ports are the ethernet ports for the system, when it fact there is a switch between these "external" ports and the devices on the JS20 system itself. The MII monitor is only able to detect link failures between the ESM and the JS20 system.

When a passthrough module is in place, the MII monitor does detect failures to the "external" port, which is then directly connected to the JS20 system.

Other concerns

The Serial Over LAN (SoL) link is established over the primary ethernet (eth0) only, therefore, any loss of link to eth0 will result in losing your SoL connection. It will not fail over with other network traffic, as the SoL system is beyond the control of the bonding driver.

It may be desirable to disable spanning tree on the switch (either the internal Ethernet Switch Module, or an external switch) to avoid fail-over delay issues when using bonding.

15. Frequently Asked Questions

1. Is it SMP safe?

Yes. The old 2.0.xx channel bonding patch was not SMP safe. The new driver was designed to be SMP safe from the start.

2. What type of cards will work with it?

Any Ethernet type cards (you can even mix cards - a Intel EtherExpress PRO/100 and a 3com 3c905b, for example). For most modes, devices need not be of the same speed.

Starting with version 3.2.1, bonding also supports Infiniband slaves in active-backup mode.

3. How many bonding devices can I have?

There is no limit.

4. How many slaves can a bonding device have?

This is limited only by the number of network interfaces Linux supports and/or the number of network cards you can place in your system.

5. What happens when a slave link dies?

If link monitoring is enabled, then the failing device will be disabled. The active-backup mode will fail over to a backup link, and other modes will ignore the failed link. The link will continue to be monitored, and should it recover, it will rejoin the bond (in whatever manner is appropriate for the mode). See the sections on High Availability and the documentation for each mode for additional information.

Link monitoring can be enabled via either the miimon or arp_interval parameters (described in the module parameters section, above). In general, miimon monitors the carrier state as sensed by the underlying network device, and the arp monitor (arp_interval) monitors connectivity to another host on the local network.

If no link monitoring is configured, the bonding driver will be unable to detect link failures, and will assume that all links are always available. This will likely result in lost packets, and a resulting degradation of performance. The precise performance loss depends upon the bonding mode and network configuration.

6. Can bonding be used for High Availability?

Yes. See the section on High Availability for details.

7. Which switches/systems does it work with?

The full answer to this depends upon the desired mode.

In the basic balance modes (balance-rr and balance-xor), it works with any system that supports etherchannel (also called trunking). Most managed switches currently available have such support, and many unmanaged switches as well.

The advanced balance modes (balance-tlb and balance-alb) do not have special switch requirements, but do need device drivers that support specific features (described in the appropriate section under module parameters, above).

In 802.3ad mode, it works with systems that support IEEE 802.3ad Dynamic Link Aggregation. Most managed and many unmanaged switches currently available support 802.3ad.

The active-backup mode should work with any Layer-II switch.

8. Where does a bonding device get its MAC address from?

When using slave devices that have fixed MAC addresses, or when the fail_over_mac option is enabled, the bonding device's MAC address is the MAC address of the active slave.

For other configurations, if not explicitly configured (with ifconfig or ip link), the MAC address of the bonding device is taken from its first slave device. This MAC address is then passed to all following slaves and remains persistent (even if the first slave is removed) until the bonding device is brought down or reconfigured.

If you wish to change the MAC address, you can set it with ifconfig or ip link:

```
# ifconfig bond0 hw ether 00:11:22:33:44:55

# ip link set bond0 address 66:77:88:99:aa:bb
```

The MAC address can be also changed by bringing down/up the device and then changing its slaves (or their order):

```
# ifconfig bond0 down ; modprobe -r bonding
# ifconfig bond0 .... up
# ifenslave bond0 eth...
```

This method will automatically take the address from the next slave that is added.

To restore your slaves' MAC addresses, you need to detach them from the bond (`ifenslave -d bond0 eth0`). The bonding driver will then restore the MAC addresses that the slaves had before they were enslaved.

16. Resources and Links

The latest version of the bonding driver can be found in the latest version of the linux kernel, found on http://kernel.org

The latest version of this document can be found in the latest kernel source (named Documentation/networking/bonding.rst).

Discussions regarding the development of the bonding driver take place on the main Linux network mailing list, hosted at vger.kernel.org. The list address is:

netdev@vger.kernel.org

The administrative interface (to subscribe or unsubscribe) can be found at:

http://vger.kernel.org/vger-lists.html#netdev