

Re: Linux Firewall/LoadBalancer

Source: <http://www.derkeiler.com/Mailing-Lists/securityfocus/security-basics/2003-11/0346.html>

From: InCisT (*InCisT_at_popsikle.net*)

Date: 11/17/03

Date: Mon, 17 Nov 2003 15:20:26 -0500

To: red <red@gato.net>

red wrote:

> All,
> I have two pipes coming into my office.
> I would like to create a redundant situation with the two networks.
> I have seen the commercial boxes that do this but I would like to do
> this with some type of Linux solution.
>
> So I would like to have one pipe running all the time , then when it
> fails , have the other take over.
>
> Any ideas?
>
>
> thanks
> -red
>
>

Redundant part (I use this on all of my HA servers):

Linux Ethernet Bonding Driver mini-howto

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>

Note :

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.

SecurityFocus BASICS: Re: Linux Firewall/LoadBalancer

For new versions of the driver, patches for older kernels and the updated userspace tools, please follow the links at the end of this file.

Table of Contents

=====

Installation
Bond Configuration
Module Parameters
Configuring Multiple Bonds
Switch Configuration
Verifying Bond Configuration
Frequently Asked Questions
High Availability
Promiscuous Sniffing notes
Limitations
Resources and Links

Installation

=====

1) Build kernel with the bonding driver

For the latest version of the bonding driver, use kernel 2.4.12 or above (otherwise you will need to apply a patch).

Configure kernel with ``make menuconfig/xconfig/config'`, and select "Bonding driver support" in the "Network device support" section. It is recommended to configure the driver as module since it is currently the only way to pass parameters to the driver and configure more than one bonding device.

Build and install the new kernel and modules.

2) Get and install the userspace tools

This version of the bonding driver requires updated ifenslave program. The original one from extreme-linux and beowulf will not work. Kernels 2.4.12 and above include the updated version of ifenslave.c in Documentation/network directory. For older kernels, please follow the links at the end of this file.

IMPORTANT!!! If you are running on Redhat 7.1 or greater, you need to be careful because `/usr/include/linux` is no longer a symbolic link to `/usr/src/linux/include/linux`. If you build ifenslave while this is true, ifenslave will appear to succeed but your bond won't work. The purpose of the `-I` option on the ifenslave compile line is to make sure it uses `/usr/src/linux/include/linux/if_bonding.h` instead of the version from `/usr/include/linux`.

SecurityFocus BASICS: Re: Linux Firewall/LoadBalancer

To install ifenslave.c, do:

```
# gcc -Wall -Wstrict-prototypes -O -I/usr/src/linux/include  
ifenslave.c -o ifenslave  
# cp ifenslave /sbin/ifenslave
```

Bond Configuration

=====

You will need to add at least the following line to /etc/modules.conf so the bonding driver will automatically load when the bond0 interface is configured. Refer to the modules.conf manual page for specific modules.conf syntax details. The Module Parameters section of this document describes each bonding driver parameter.

```
alias bond0 bonding
```

Use standard distribution techniques to define the bond0 network interface. For example, on modern Red Hat distributions, create an ifcfg-bond0 file in the /etc/sysconfig/network-scripts directory that resembles the following:

```
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
```

(use appropriate values for your network above)

All interfaces that are part of a bond should have SLAVE and MASTER definitions. For example, in the case of Red Hat, if you wish to make eth0 and eth1 a part of the bonding interface bond0, their config files (ifcfg-eth0 and ifcfg-eth1) should resemble the following:

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

Use DEVICE=eth1 in the ifcfg-eth1 config file. If you configure a second bonding interface (bond1), use MASTER=bond1 in the config file to make the network interface be a slave of bond1.

SecurityFocus BASICS: Re: Linux Firewall/LoadBalancer

Restart the networking subsystem or just bring up the bonding device if your administration tools allow it. Otherwise, reboot. On Red Hat distros you can issue ``ifup bond0'` or ``/etc/rc.d/init.d/network restart'`.

If the administration tools of your distribution do not support master/slave notation in configuring network interfaces, you will need to manually configure the bonding device with the following commands:

```
# /sbin/ifconfig bond0 192.168.1.1 netmask 255.255.255.0 \  
broadcast 192.168.1.255 up  
  
# /sbin/ifenslave bond0 eth0  
# /sbin/ifenslave bond0 eth1
```

(use appropriate values for your network above)

You can then create a script containing these commands and place it in the appropriate rc directory.

If you specifically need all network drivers loaded before the bonding driver, adding the following line to `modules.conf` will cause the network driver for `eth0` and `eth1` to be loaded before the bonding driver.

```
probeall bond0 eth0 eth1 bonding
```

Be careful not to reference `bond0` itself at the end of the line, or `modprobe` will die in an endless recursive loop.

To have device characteristics (such as MTU size) propagate to slave devices, set the bond characteristics before enslaving the device. The characteristics are propagated during the `enslave` process.

If running SNMP agents, the bonding driver should be loaded before any network drivers participating in a bond. This requirement is due to the 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.

Module Parameters

=====

Optional parameters for the bonding driver can be supplied as command line arguments to the insmod command. Typically, these parameters are specified in the file /etc/modules.conf (see the manual page for modules.conf). 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.

max_bonds

SecurityFocus BASICS: Re: Linux Firewall/LoadBalancer

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.

mode

Specifies one of four bonding policies. The default is round-robin (`balance-rr`). Possible values are (you can use either the text or numeric option):

`balance-rr` or 0

Round-robin policy: Transmit in a 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. This mode provides fault tolerance.

`balance-xor` or 2

XOR policy: Transmit based on [(source MAC address XOR'd with destination MAC address) modulo slave count]. This selects the same slave for each destination MAC address. This mode provides load balancing and fault tolerance.

`broadcast` or 3

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

miimon

Specifies the frequency in milli-seconds that MII link monitoring will occur. A value of zero disables MII link monitoring. A value of 100 is a good starting point. See High Availability section for additional information. The default value is 0.

use_carrier

Specifies whether or not `miimon` should use MII or `ETHTOOL` `ioctl`s vs. `netif_carrier_ok()` to determine the link status. The MII or `ETHTOOL` `ioctl`s 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`. This is because the default state for `netif_carrier` is "carrier on." In this case, disabling `use_carrier` 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.

`downdelay`

Specifies the delay time in milli-seconds to disable a link after a link failure has been detected. This should be a multiple of `miimon` value, otherwise the value will be rounded. The default value is 0.

`updelay`

Specifies the delay time in milli-seconds to enable a link after a link up status has been detected. This should be a multiple of `miimon` value, otherwise the value will be rounded. The default value is 0.

`arp_interval`

Specifies the ARP monitoring frequency in milli-seconds. If ARP monitoring is used in a load-balancing mode (mode 0 or 2), the switch should be configured in a mode that evenly distributes packets across all links – such as round-robin. 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 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 needs to be given for ARP monitoring to work. The maximum number of targets that can be specified is set at 16.

primary

A string (eth0, eth2, etc) to equate to a primary device. If this value is entered, and the device is on-line, it will be used first as the output media. Only when this device is off-line, will alternate devices be used. Otherwise, once a failover is detected and a new default output is chosen, it will remain the output media until it too fails. This is useful when one slave was preferred over another, i.e. when one slave is 1000Mbps and another is 100Mbps. If the 1000Mbps slave fails and is later restored, it may be preferred the faster slave gracefully become the active slave – without deliberately failing the 100Mbps slave. Specifying a primary is only valid in active-backup mode.

multicast

Option specifying the mode of operation for multicast support.
Possible values are:

disabled or 0

Disabled (no multicast support)

active or 1

Enabled on active slave only, useful in active-backup mode

all or 2

Enabled on all slaves, this is the default

Configuring Multiple Bonds

=====

If several bonding interfaces are required, the driver must be loaded multiple times. For example, to configure two bonding interfaces with link monitoring performed every 100 milli-seconds, the /etc/conf.modules should resemble the following:

alias bond0 bonding
alias bond1 bonding

```
options bond0 miimon=100
options bond1 -o bonding1 miimon=100
```

Configuring Multiple ARP Targets

=====

While ARP monitoring can be done with just one target, it can be usefull 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) would increase the reliability of the ARP monitoring. Multiple ARP targets must be seperated 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
```

Potential Problems When Using ARP Monitor

=====

1. Driver support

The ARP monitor relies on the network device driver to maintain two statistics: the last receive time (dev->last_rx), and the last transmit time (dev->trans_start). If the network device driver does not update one or both of these, then the typical result will be that, upon startup, all links in the bond will immediately be declared down, and remain that way. A network monitoring tool (tcpdump, e.g.) will show ARP requests and replies being sent and received on the bonding device.

The possible resolutions for this are to (a) fix the device driver, or (b) discontinue the ARP monitor (using miimon as an alternative, for example).

2. Adventures in Routing

When bonding is set up with the ARP monitor, it is important that the slave devices not have routes that supercede 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
```

In this case, the ARP monitor (and ARP itself) may become confused, because ARP requests 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. This will likely still update the receive/transmit times in the driver, but will lose packets.

The resolution 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 supercede routes of their master. This should generally be the case, but unusual configurations or errant manual or automatic static route additions may cause trouble.

Switch Configuration

=====

While the switch does not need to be configured when the active-backup policy is used (mode=1), it does need to be configured for the round-robin, XOR, and broadcast policies (mode=0, mode=2, and mode=3).

Verifying Bond Configuration

=====

1) Bonding information files

The bonding driver information files reside in the /proc/net/bond* directories.

Sample contents of /proc/net/bond0/info after the driver is loaded with parameters of mode=0 and miimon=1000 is shown below.

```
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

2) Network verification

The network configuration can be verified using the `ifconfig` command. 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`.

```
[root]# /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
       inet addr:XXX.XXX.XXX.YYY Bcast:XXX.XXX.XXX.255
Mask:255.255.252.0
       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
       inet addr:XXX.XXX.XXX.YYY Bcast:XXX.XXX.XXX.255
Mask:255.255.252.0
       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
```

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). You can even bond together Gigabit Ethernet cards!

3. How many bonding devices can I have?

One for each module you load. See section on Module Parameters for how to accomplish this.

4. How many slaves can a bonding device have?

Limited 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 your ethernet cards support MII or ETHTOOL link status monitoring and the MII monitoring has been enabled in the driver (see description

of module parameters), there will be no adverse consequences. This release of the bonding driver knows how to get the MII information and enables or disables its slaves according to their link status. See section on High Availability for additional information.

For ethernet cards not supporting MII status, the arp_interval and arp_ip_target parameters must be specified for bonding to work correctly. If packets have not been sent or received during the specified arp_interval duration, an ARP request is sent to the targets

to generate send and receive traffic. If after this interval, either the successful send and/or receive count has not incremented, the next slave in the sequence will become the active slave.

If neither mii_monitor and arp_interval is configured, the bonding driver will not handle this situation very well. The driver will continue to send packets but some packets will be lost. Retransmits will cause serious degradation of performance (in the case when one of two slave links fails, 50% packets will be lost, which is a serious problem for both TCP and UDP).

6. Can bonding be used for High Availability?

Yes, if you use MII monitoring and ALL your cards support MII link status reporting. See section on High Availability for more information.

7. Which switches/systems does it work with?

SecurityFocus BASICS: Re: Linux Firewall/LoadBalancer

In round-robin and XOR mode, it works with systems that support trunking:

- * Cisco 5500 series (look for EtherChannel support).
- * SunTrunking software.
- * Alteon AceDirector switches / WebOS (use Trunks).
- * BayStack Switches (trunks must be explicitly configured). Stackable models (450) can define trunks between ports on different physical units.
- * Linux bonding, of course !

In active-backup mode, it should work with any Layer-II switch.

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

If not explicitly configured with `ifconfig`, 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`:

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

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 will be added.

To restore your slaves' MAC addresses, you need to detach them from the bond (`ifenslave -d bond0 eth0`), set them down (`ifconfig eth0 down`), unload the drivers (`rmmod 3c59x`, for example) and reload them to get the MAC addresses from their eeproms. If the driver is shared by several devices, you need to turn them all down. Another solution is to look for the MAC address at boot time (`dmesg` or `tail /var/log/messages`) and to reset it by hand with `ifconfig` :

```
# ifconfig eth0 down
# ifconfig eth0 hw ether 00:20:40:60:80:A0
```

9. Which transmit polices can be used?

Round-robin, based on the order of enslaving, the output device is selected base on the next available slave. Regardless of

the source and/or destination of the packet.

Active-backup policy that ensures that one and only one device will transmit at any given moment. Active-backup policy is useful for implementing high availability solutions using two hubs (see section on High Availability).

XOR, based on (src hw addr XOR dst hw addr) % slave count. This policy selects the same slave for each destination hw address.

Broadcast policy transmits everything on all slave interfaces.

High Availability

=====

To implement high availability using the bonding driver, the driver needs to be compiled as a module, because currently it is the only way to pass parameters to the driver. This may change in the future.

High availability is achieved by using MII or ETHTOOL status reporting. You need to verify that all your interfaces support MII or ETHTOOL link status reporting. On Linux kernel 2.2.17, all the 100 Mbps capable drivers and yellowfin gigabit driver support MII. To determine if ETHTOOL link reporting is available for interface eth0, type "ethtool eth0" and the "Link detected:" line should contain the correct link status. If your system has an interface that does not support MII or ETHTOOL status reporting, a failure of its link will not be detected! A message indicating MII and ETHTOOL is not supported by a network driver is logged when the bonding driver is loaded with a non-zero miimon value.

The bonding driver can regularly check all its slaves links using the ETHTOOL IOCTL (ETHTOOL_GLINK command) or by checking the MII status registers. The check interval is specified by the module argument "miimon" (MII monitoring).

It takes an integer that represents the checking time in milliseconds. It should not come too close to (1000/HZ) (10 milli-seconds on i386) because it may then reduce the system interactivity. A value of 100 seems to be a good starting point. It means that a dead link will be detected at most 100 milli-seconds after it goes down.

Example:

SecurityFocus BASICS: Re: Linux Firewall/LoadBalancer

```
# modprobe bonding miimon=100
```

Or, put the following lines in `/etc/modules.conf`:

```
alias bond0 bonding
options bond0 miimon=100
```

There are currently two policies for high availability. They are dependent on whether:

- a) hosts are connected to a single host or switch that support trunking
- b) hosts are connected to several different switches or a single switch that does not support trunking

1) High Availability on a single switch or host – load balancing

It is the easiest to set up and to understand. Simply configure the remote equipment (host or switch) to aggregate traffic over several ports (Trunk, EtherChannel, etc.) and configure the bonding interfaces. If the module has been loaded with the proper MII option, it will work automatically. You can then try to remove and restore different links and see in your logs what the driver detects. When testing, you may encounter problems on some buggy switches that disable the trunk for a long time if all ports in a trunk go down. This is not Linux, but really the switch (reboot it to ensure).

Example 1 : host to host at twice the speed

```
+-----+ +-----+
| |eth0 eth0| |
| Host A +-----+ Host B |
| +-----+ |
| |eth1 eth1| |
+-----+ +-----+
```

On each host :

```
# modprobe bonding miimon=100
# ifconfig bond0 addr
# ifenslave bond0 eth0 eth1
```

Example 2 : host to switch at twice the speed

```
+-----+ +-----+
| |eth0 port1| |
| Host A +-----+ switch |
| +-----+ |
| |eth1 port2| |
+-----+ +-----+
```

SecurityFocus BASICS: Re: Linux Firewall/LoadBalancer

On host A : On the switch :

```
# modprobe bonding miimon=100 # set up a trunk on port1
# ifconfig bond0 addr and port2
# ifenslave bond0 eth0 eth1
```

2) High Availability on two or more switches (or a single switch without trunking support)

This mode is more problematic because it relies on the fact that there are multiple ports and the host's MAC address should be visible on one port only to avoid confusing the switches.

If you need to know which interface is the active one, and which ones are backup, use ifconfig. All backup interfaces have the NOARP flag set.

To use this mode, pass "mode=1" to the module at load time :

```
# modprobe bonding miimon=100 mode=active-backup
```

or:

```
# modprobe bonding miimon=100 mode=1
```

Or, put in your /etc/modules.conf :

```
alias bond0 bonding
options bond0 miimon=100 mode=active-backup
```

Example 1: Using multiple host and multiple switches to build a "no single point of failure" solution.

```
  ||
  |port3 port3|
+-----+-----+ +-----+-----+
| |port7 ISL port7| |
| switch A +-----+ switch B |
| +-----+ |
| |port8 port8| |
+-----+-----+ +-----+-----+
port2||port1 port1||port2
  || +-----+ ||
  | +-----+ host1 +-----+ |
  | eth0 +-----+ eth1 |
  ||
  | +-----+ |
  +-----+ host2 +-----+
      eth0 +-----+ eth1
```

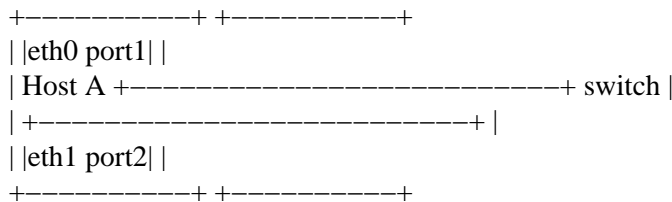
In this configuration, there is an ISL – Inter Switch Link (could be a trunk), several servers (host1, host2 ...) attached to both switches each, and

one or more ports to the outside world (port3...). One an only one slave on each host is active at a time, while all links are still monitored (the system can detect a failure of active and backup links).

Each time a host changes its active interface, it sticks to the new one until it goes down. In this example, the hosts are negligibly affected by the expiration time of the switches' forwarding tables.

If host1 and host2 have the same functionality and are used in load balancing by another external mechanism, it is good to have host1's active interface connected to one switch and host2's to the other. Such system will survive a failure of a single host, cable, or switch. The worst thing that may happen in the case of a switch failure is that half of the hosts will be temporarily unreachable until the other switch expires its tables.

Example 2: Using multiple ethernet cards connected to a switch to configure NIC failover (switch is not required to support trunking).



```

On host A : On the switch :
# modprobe bonding miimon=100 mode=1 # (optional) minimize the
time
# ifconfig bond0 addr # for table expiration
# ifenslave bond0 eth0 eth1
    
```

Each time the host changes its active interface, it sticks to the new one until it goes down. In this example, the host is strongly affected by the expiration time of the switch forwarding table.

3) Adapting to your switches' timing

 If your switches take a long time to go into backup mode, it may be desirable not to activate a backup interface immediately after a link goes down. It is possible to delay the moment at which a link will be completely disabled by passing the module parameter "downdelay" (in milliseconds, must be a multiple of miimon).

SecurityFocus BASICS: Re: Linux Firewall/LoadBalancer

When a switch reboots, it is possible that its ports report "link up" status before they become usable. This could fool a bond device by causing it to use some ports that are not ready yet. It is possible to delay the moment at which an active link will be reused by passing the module parameter "updelay" (in milliseconds, must be a multiple of miimon).

A similar situation can occur when a host re-negotiates a lost link with the switch (a case of cable replacement).

A special case is when a bonding interface has lost all slave links. Then the driver will immediately reuse the first link that goes up, even if updelay parameter was specified. (If there are slave interfaces in the "updelay" state, the interface that first went into that state will be immediately reused.) This allows to reduce down-time if the value of updelay has been overestimated.

Examples :

```
# modprobe bonding miimon=100 mode=1 downdelay=2000 updelay=5000
# modprobe bonding miimon=100 mode=balance-rr downdelay=0 updelay=5000
```

Promiscuous Sniffing notes

=====

If you wish to bond channels together for a network sniffing application --- you wish to run tcpdump, or ethereal, or an IDS like snort, with its input aggregated from multiple interfaces using the bonding driver ---- then you need to handle the Promiscuous interface setting by hand. Specifically, when you "ifconfig bond0 up" you must add the promisc flag there; it will be propagated down to the slave interfaces at ifenslave time; a full example might look like:

```
grep bond0 /etc/modules.conf || echo alias bond0 bonding
>/etc/modules.conf
ifconfig bond0 promisc up
for if in eth1 eth2 ...;do
    ifconfig $if up
    ifenslave bond0 $if
done
snort ... -i bond0 ...
```

Ifenslave also wants to propagate addresses from interface to interface, appropriately for its design functions in HA and channel capacity aggregating; but it works fine for unnumbered interfaces; just ignore all the warnings it emits.

Limitations

=====

SecurityFocus BASICS: Re: Linux Firewall/LoadBalancer

The main limitations are :

- only the link status is monitored. If the switch on the other side is partially down (e.g. doesn't forward anymore, but the link is OK), the link won't be disabled. Another way to check for a dead link could be to count incoming frames on a heavily loaded host. This is not applicable to small servers, but may be useful when the front switches send multicast information on their links (e.g. VRRP), or even health-check the servers.
Use the `arp_interval/arp_ip_target` parameters to count incoming/outgoing frames.
- A Transmit Load Balancing policy is not currently available. This mode allows every slave in the bond to transmit while only one receives. If the "receiving" slave fails, another slave takes over the MAC address of the failed receiving slave.

Resources and Links

=====

Current development on this driver is posted to:

- <http://www.sourceforge.net/projects/bonding/>

Donald Becker's Ethernet Drivers and diag programs may be found at :

- <http://www.scyld.com/network/>

You will also find a lot of information regarding Ethernet, NWay, MII, etc. at www.scyld.com.

For new versions of the driver, patches for older kernels and the updated userspace tools, take a look at Willy Tarreau's site :

- <http://wtarreau.free.fr/pub/bonding/>
- <http://www-miaif.lip6.fr/willy/pub/bonding/>

To get latest informations about Linux Kernel development, please consult the Linux Kernel Mailing List Archives at :

<http://boudicca.tux.org/hypermail/linux-kernel/latest/>

-- END --

Forum Systems PRESIDIO: PGP / XML GATEWAY APPLIANCE

The Presidio integrates PGP data encryption and XML Web Services security to simplify the management and deployment of PGP and reduce overall PGP costs by up to 80%.

FREE WHITEPAPER & 30 Day Trial –

Re: Linux Firewall/LoadBalancer

SecurityFocus BASICS: Re: Linux Firewall/LoadBalancer

http://www.securityfocus.com/sponsor/ForumSystems_security-basics_031027
