Package ADVANCED_NETWORKING
Version 3.10.4

Frank Meyer
email: frank@fli4l.de

the fli4l-Team
email: team@fli4l.de

October 25, 2015

Contents

1 Documentation of the package ADVANCED_NETWORKING

1.1 Advanced Networking

The package 'advanced networking' provides bonding and bridging capabilities for the fli4l-router. EBTables (http://ebtables.sourceforge.net/) support can be enabled as well. This allows to build a transparent packet filter. To all options of the advanced_networking package generally applies:

This package is only for users with profund knowledge about networks and routing.

Very unusual problems can appear especially using EBTables without perfectly knowing the diverse operational modes of layer 2 and 3. Some filtering rules of the packet filter will work completely different with EBTables support enabled.

1.1.1 Broadcast Relay - Forwarding of IP Broadcasts

Using a Broadcast Relay, IP broadcasts can surpass interface boundaries. This is necessary for applications which determine network devices using broadcasts (eg QNAP Finder). Broadcasts are normally not passed across network boundaries by the router. This problem can be circumvented by using a broadcast relay.

Within a Broadcast Relay broadcasts are always forwarded to all connected interfaces. This means that setting up a second Broadcast Relay with interfaces swapped is not necessary. In addition, multiple broadcast relays including the same interface are not allowed.

OPT_BCRELAY Broadcast Forwarding

Default: `OPT_BCRELAY='no'`

Setting `'yes'` here activates the Broadcast Relay package. Specifying `'no'` deactivates the Broadcast Relay package completely.

BCRELAY_N Default: `BCRELAY_N='0'`

The number of Broadcast Relays to configure.

BCRELAY_x_IF_N Default: `BCRELAY_x_IF_N='1'`

Number of interfaces assigned to this broadcast relay.

BCRELAY_x_IF_x Default: `BCRELAY_x_IF_x="` Name of the interface assigned to this broadcast relay.

For illustration an example follows where the computer with the application (eg QNAP Finder) is located in the internal network (connected to `eth0`) and the NAS is in a different network (connected to `eth1`).

```
OPT_BCRELAY='yes'
BCRELAY_N='1'
BCRELAY_1_IF_N='2'
BCRELAY_1_IF_1='eth0'
BCRELAY_1_IF_2='eth1'
```

1.1.2 Bonding - Combining Several Network Interface Cards In One Link

Bonding refers to joining at least two network interface cards into one link. The cards even may be of different type (ie 3Com and Intel) or speed (ie 10 Mbit/s or 100 Mbit/s). You can either connect linux computers directly or connect to a network switch using bonding. In this way a 200 Mbit/s full duplex connection from a flil4-router to a switch can be used without much effort. Everyone interested in using bonding should have read the documentation in the kernel directory (bonding.txt). The names of the bonding settings largely correspond to the names used there.

OPT_BONDING_DEV Default: `OPT_BONDING_DEV='no'`

`'yes'` activates the bonding package, `'no'` deativates the bonding package completely.

BONDING_DEV_N Default: `BONDING_DEV_N='0'`

Number of bonding devices to be configured.

BONDING_DEV_x_DEVNAME Default: `BONDING_DEV_x_DEVNAME="`

Name of the bonding device to be created. It should consist of the prefix 'bond' and a trailing number with out a leading '0'. The numbers of the bonding devices don't have to start with '0' and need not be consecutive. Possible values could be 'bond0', 'bond8' or 'bond99'.

BONDING_DEV_x_MODE Default: `BONDING_DEV_x_MODE="`

Specifies the bonding method. Default is round-robin 'balance-rr'. Possible values are listed below:

balance-rr Round-robin method: Submit sequentially over all slaves from the first to the last. This method provides both load balancing and fault tolerance.

active-backup Active backup: Only one slave in the bond is active. The other slaves are activated only when the active slave fails. The MAC address of the bond is only visible on one port (network adapter) so it does not confuse the switch. This mode provides fault tolerance.

balance-xor XOR method: Submit based on the formula [(Source-MAC-address XOR destination-MAC-address) modulo the number of slaves]. This ensures that the same slave always is used for the same destination-MAC-address. This method provides both load balancing and fault tolerance.

broadcast Broadcast method: Transmits everything on all slave devices. This mode provides fault tolerance.

802.3ad IEEE 802.3ad dynamic link aggregation. Creates aggregation groups that share the same speed and duplex settings. Transmits on all slaves in the active aggregator.

Requirements:

- ethtool support in the base device driver to retrieve speed and duplex status for each device.
- a switch that supports dynamic IEEE 802.3ad connection aggregation.

balance-tlb Adaptive load balancing for outgoing data: channel bonding that does not need any special features in the switch. The outgoing network traffic is distributed on each slave according to the current load. Incoming network traffic is received by the current slave. If the receiving slave fails, another slave takes over the MAC address of the slave gone down.

Requirements:

- ethtool support in the base device driver to retrieve speed and duplex status for each device.

balance-alb Adaptive load balancing: includes both balance-tlb, and inbound load balancing (rlb) for IPV4 traffic and needs no special requirements on the Switch. Load Balancing for incoming traffic is achieved through ARP requests. The bonding driver catches ARP responses from the server on their way outside and overrides the source hardware address with the unique hardware address of a slave in the bond. This way different clients use different hardware addresses for the server.

Incoming traffic from connections created by the server will also be balanced. If the server sends ARP requests, the bonding driver copies and stores the client IP from the ARP. At the time the ARP response of the client arrives the bonding driver determines its hardware address and creates an ARP reply to this client assigning a client in the bond to it. A problematic effect of ARP arrangements for load balancing is that every time an ARP request is sent the hardware address of the bond is used. Clients learn the hardware address of the bond and the incoming traffic on the current slave collapses. This fact is countered in a way that updates (ARP Replies) to all clients will be sent to their respective hardware addresses so that the traffic is divided again. Incoming traffic will be newly allocated even when a new slave is added to the bonding or an inactive slave is re-activated. The receiving load is distributed sequentially (round robin) in the group of the slave with the largest network speed in the bond.

When a connection is restored or a new slave joins the bond incoming traffic will be distributed anew to all active Slaves in the bond by sending ARP replies with the selected MAC addresses to each client. The parameter 'updelay' must be set to a value greater than or equal to the forwarding delay of the switch in order to avoid blocking of ARP responses to clients.

Requirements:

- ethtool support in the base device driver to retrieve speed and duplex status for each device.
- support in the base device driver to set the hardware address even when the device is open. This is neccessary for granting that at every time at least one slave in the bond is carrying the hardware address of the bond (curr_active_slave) although every slave in the bond has its own unique hardware address. If curr_active_slave fails its hardware address will simply be replaced with a new one.

BONDING_DEV_x_DEV_N Default: `BONDING_DEV_x_DEV_N='0'`

Specifies the number of physical devices the bond consists of. E.g. for a bond between 'eth0' and 'eth1' (two eth-devices) '2' has to be entered.

BONDING_DEV_x_DEV_x Default: `BONDING_DEV_x_DEV_x=""`

The name of a physical device which belongs to this bonding device. An example would be the value 'eth0'. Please note that a physical device that you use for a bond can't be used for anything else. So you can't use it additionally for a DSL modem, a bridge, a VLAN or inclusion in base.txt.

BONDING_DEV_x_MAC Default: `BONDING_DEV_x_MAC=""`

This setting is optional and can also be completely omitted.

A bonding device defaults to the MAC address of the first physical device which is used for bonding. If you do not want this it is possible to specify a MAC address the bonding device should use here.

BONDING_DEV_x_MIIMON Default: `BONDING_DEV_x_MIIMON='100'`

This setting is optional and can also be completely omitted.

Specifies the interval (in milliseconds) in which the individual connections of a bonding device are checked for their link status. The link status of each physical device in the bond will be checked every x milliseconds. Setting this to '0' will disable the miimon monitoring.

BONDING_DEV_x_USE_CARRIER Default: `BONDING_DEV_x_USE_CARRIER='yes'`

This setting is optional and can also be completely omitted.

If the link status check by miimon (see above) is enabled this setting can specify the function performing the check.

- `'yes'`: netif _carrier_ok() function
- `'no`: direct calls to MII or ethtool ioctl() System calls

The netif_carrier_ok() method is more efficient, but not all drivers do support this method.

BONDING_DEV_x_UPDELAY Default: `BONDING_DEV_x_UPDELAY='0'`

This setting is optional and can also be completely omitted.

The value of this setting multiplied by the setting of `BONDING_DEV_x_MIIMON` specifies the time a in which a connection of bonding devices is activated after the corresponding link (for example 'eth0') is up. This way the connection of the bonding device is activated until the link status switches to "not connected".

BONDING_DEV_x_DOWNDELAY Default: `BONDING_DEV_x_DOWNDELAY='0'`

This setting is optional and can also be completely omitted.

The value of this setting multiplied by the setting of `BONDING_DEV_x_MIIMON` specifies the time a in which a connection of bonding devices is deactivated if the appropriate link (iE an eth-device) fails. This will deactivate the connection of a bonding device temporarily until the link status is back to 'active'.

BONDING_DEV_x_LACP_RATE Default: `BONDING_DEV_x_LACP_RATE='slow'`

This setting is optional and can also be completely omitted.

Specifiy how often link informations are exchanged between the link partners (for example a switch or another linux PC) if `BONDING_DEV_x_MODE="` is set to '802.3ad'.

- 'slow': every 30 seconds
- 'fast': each second.

BONDING_DEV_x_PRIMARY Default: `BONDING_DEV_x_PRIMARY="`

This setting is optional and can also be completely omitted.

Specify primary output device if mode is set to 'active-backup'. This is useful if the various devices have different speeds. Provide a string (for example 'eth0') for the device to be used primarily. If a value is entered and the device is online it will be used as the first output medium. Only if the device is offline another device will be used. If a failure is detected a new standard output medium will be chosen. This comes in handy if one slave has priority over another, for example if a slave is faster than another (1000 Mbit/s versus 100 Mbit/s). If the 1000 Mbit/s slave fails and later gets back up it may be of advantage to set the faster slave active again without having to cause a fail of the 100 Mbit/s slave artificially (for example by pulling the plug).

BONDING_DEV_x_ARP_INTERVAL Default: `BONDING_DEV_x_ARP_INTERVAL='0'`

This setting is optional and can also be completely omitted.

The interval in which IP-addresses specified in `BONDING_DEV_x_ARP_IP_TARGET_x` are checked by using their ARP responses (in milliseconds). If ARP monitoring is used in load-balancing mode (mode 0 or 2) the switch should be adjusted to distribute packets to all connections equally (for example round robin). If the switch is set to distribute the packets according to the XOR method all responses of the ARP targets will arrive on the same connection which could cause failure for all team members. ARP monitoring should not be combined with miimon. Passing '0' will disable ARP monitoring.

BONDING_DEV_x_ARP_IP_TARGET_N Default: `BONDING_DEV_x_ARP_IP_TARGET_N="`

This setting is optional and can also be completely omitted.

The number of IP-addresses which are used for ARP checking. A maximum of 16 IP-addresses can be checked.

BONDING_DEV_x_ARP_IP_TARGET_x Default: `BONDING_DEV_x_ARP_IP_TARGET_x="`

This setting is optional and can also be completely omitted.

If `BONDING_DEV_x_ARP_INTERVAL` is > 0, specify one IP address which is used as the target for ARP requests to evaluate the quality of the connection. Enter values using format 'ddd.ddd.ddd.ddd'. To get ARP monitoring to work at least one IP address has to be given here.

1.1.3 VLAN - 802.1Q Support

Support for 802.1Q VLAN is reasonable only in conjunction with using appropriate switches. Port-based VLAN switches are *not* suitable. A general introduction to the subject VLAN can be found at http://en.wikipedia.org/wiki/IEEE_802.1Q. At http://de.wikipedia.org/wiki/VLAN some additional information can be found.

Please note that not any network card can handle VLANs. Some can not handle VLANs at all, others require a matching MTU and few cards work without any problems. The author of the advanced_networking package uses Intel network cards with the 'e100' driver without any problem. MTU adjustment is not necessary. 3COM's '3c59x' driver requires MTU adjustment to 1496 otherwise the card won't work correctly. The 'starfire' driver does not work properly if a VLAN device is added to a bridge. In this case no packets can be received. Those who want to work with VLANs should ensure that the respective Linux NIC drivers support VLANs correctly.

OPT_VLAN_DEV Default: `OPT_VLAN_DEV='no'`

`'yes'` activates the VLAN package, `'no'` deactivates it.

VLAN_DEV_N Default: `VLAN_DEV_N="`

Number of VLAN devices to configure.

VLAN_DEV_x_DEV Default: `VLAN_DEV_x_DEV="`

Name of the device connected to a VLAN capable switch (iE `'eth0'`, `'br1'`, `'eth2'`...).

VLAN_DEV_x_VID Default: `VLAN_DEV_x_VID="`

The VLAN ID for which the appropriate VLAN device should be created. The name of the VLAN device consists of the prefix 'ethX' and the attached VLAN ID (without leading '0'). For example '42' creates a VLAN device `'eth0.42'` on the fli4l-router.

VLAN devices on the fli4l-router are always named '<device>.<vid>'. So if you have an eth-device connected to a VLAN-capable switch and you want to use VLANs 10, 11 and 23 on the fli4l-router you have to configure 3 VLAN devices with the eth-device as `VLAN_DEV_x_DEV='ethX'` and the respective VLAN ID in `VLAN_DEV_x_VID="`. Example:

```
OPT_VLAN_DEV='yes'
VLAN_DEV_N='3'
VLAN_DEV_1_DEV='eth0'
VLAN_DEV_1_VID='10' # will create device: eth0.10
VLAN_DEV_2_DEV='eth0'
VLAN_DEV_2_VID='11' # will create device: eth0.11
VLAN_DEV_3_DEV='eth0'
VLAN_DEV_3_VID='23' # will create device: eth0.23
```

Please always remember to check the MTU of all units involved. Caused by the VLAN header the frames will be 4 bytes longer. If necessary the MTU must be changed to 1496 on the devices.

1.1.4 Device MTU - Adjusting MTU Values

In rare circumstances it may be necessary to adjust the MTU of a device. E.G. some not 100% VLAN-compatible network cards need to adjust the MTU. Please remember that few network cards are capable of processing Ethernet frames larger than the 1500 bytes!

DEV_MTU_N Default: `DEV_MTU_N="`

This setting is optional and can also be completely omitted.

Number of devices to change their MTU settings.

DEV_MTU_x Default: `DEV_MTU_x="`

This setting is optional and can also be completely omitted.

Name of the device to change its MTU followed by the MTU to be set. Both statements have to be separated by a space. To set a MTU of `'1496'` for `'eth0'` enter the following:

```
DEV_MTU_N='1'
DEV_MTU_1='eth0 1496'
```

1.1.5 BRIDGE - Ethernet Bridging for fli4l

This is a full-fledged ethernet-bridge using spanning tree protocol on demand. For the user the Computer seems to work as a layer 3 switch on configured ports.
Further information on bridging can be found here:

- Homepage of the Linux Bridging Project:
  http://bridge.sourceforge.net/

- The detailed and authoritative description of the bridging standards:
  http://standards.ieee.org/getieee802/download/802.1D-2004.pdf.
  (Mainly informations from page 153 on are interesting. Please note that the Linux bridging code is working according to standards from 1998, allowing only 16 bit Values for pathcost as an example.)

- Calculation of different timing values for the spanning tree protocol:
  http://www.dista.de/netstpclc.htm

- See how STP is working by looking at some nice examples:
  http://web.archive.org/web/20060114052801/http://www.zyxel.com/support/supportnote/ves1012/app/stp.htm

OPT_BRIDGE_DEV Default: `OPT_BRIDGE_DEV='no'`

`'yes'` activates the bridge package, `'no'` deactivates it.

BRIDGE_DEV_BOOTDELAY Default: `BRIDGE_DEV_BOOTDELAY='yes'`

This setting is optional and can also be completely omitted.

As a bridge needs at least 2 × `BRIDGE_DEV_x_FORWARD_DELAY` in seconds to become active this period has to be waited if devices are needed at the startup of the fli4l-router. As an example consider sending syslog messages or dialing in via DSL. If the entry is set to `'yes'` 2 × `BRIDGE_DEV_x_FORWARD_DELAY` is waited automatically. If the bridges are not required at startup-time `'no'` should be set to accelerate the startup process of fli4l router.

BRIDGE_DEV_N Default: `BRIDGE_DEV_N='1'`

The number of independent bridges. Each bridge has to be considered completely isolated. This applies in particular for the setting of `BRIDGE_DEV_x_STP`. There will be created one virtual device by the name of `'br<nummer>'` per bridge.

BRIDGE_DEV_x_NAME Default: `BRIDGE_DEV_x_NAME="`

The symbolic name of the bridge. This name can be used by other packages in order to use the bridge regardless of its device name.

BRIDGE_DEV_x_DEVNAME Default: `BRIDGE_DEV_x_DEVNAME="`

Each bridge device needs a name in the form of `'br<number>'`. <number> can be a number between '0' and '99' without leading '0'. Possible entries could be `'br0'`, `'br9'` or `'br42'`. Names can be chosen arbitrary, the first bridge may be `'br3'` and the second `'br0'`.

BRIDGE_DEV_x_DEV_N Default: `BRIDGE_DEV_x_DEV_N='0'`

How many network devices belong to the bridge? The count of devices that should be connected to the bridge. It can even be `'0'` if the bridge is only a placeholder for an IP-address that should be taken over by a VPN-tunnel connected to the bridge.

BRIDGE_DEV_x_DEV_x_DEV Specifies which device can be connected to the bridge. You can fill in an eth-device (ie `'eth0'`), a bonding device (iE `'bond0'`) or also a VLAN-device (iE `'vlan11'`). A device connected here may not be used in other places and is not allowed to get an IP-address assigned.

```
BRIDGE_DEV_1_DEV_N='3'
BRIDGE_DEV_1_DEV_1_DEV='eth0.11' #VLAN 11 on eth0
BRIDGE_DEV_1_DEV_2_DEV='eth2'
BRIDGE_DEV_1_DEV_3_DEV='bond0'
```

BRIDGE_DEV_x_AGING Default: `BRIDGE_DEV_x_AGING='300'`

This setting is optional and can also be completely omitted.

Specifies the time after which old entries in the bridges' MAC table will be deleted. If in this amount of time in seconds no data is received or transmitted by the computer with the network card the corresponding MAC address will be deleted in the bridges' MAC table.

BRIDGE_DEV_x_GARBAGE_COLLECTION_INTERVAL

Default: `BRIDGE_DEV_x_GARBAGE_COLLECTION_INTERVAL='4'`

This setting is optional and can also be completely omitted.

Specifies the time after which „garbage collection“ will be done. All dynamic entries will be checked. Entries not longer valid and outdated will get deleted. In particular old invalid connections will be deleted.

BRIDGE_DEV_x_STP Default: `BRIDGE_DEV_x_STP='no'`

This setting is optional and can also be completely omitted.

Spanning tree protocol allows to manage multiple connections to different switches. This results in redundancy ensuring network functionality in case of line failures. Without the use of STP redundant lines between switches aren't possible and networking may fail. STP tries to use the fastest connection between two switches. This way even connections with different speeds are reasonable. You may iE use a 1 Gbit/s connection as main and a second 100 Mbit/s as a fallback.

A good source of background informations can be found here:
http://en.wikipedia.org/wiki/Spanning_Tree_Protocol.

BRIDGE_DEV_x_PRIORITY Default: `BRIDGE_DEV_x_PRIORITY="`

This setting is optional and can also be completely omitted.

Only valid if `BRIDGE_DEV_x_STP='yes'` is set!

Which priority has this bridge? The bridge with the lowest priority wins the main bridge election. Each bridge should have a different priority. Please note that the bridge with the lowest priority should also have the biggest available bandwith because in addition to the complete data traffic control packets will be sent by it every 2 seconds. (See also: `BRIDGE_DEV_x_HELLO`)

Valid Values are from `'0'` to `'61440'` in steps of 4096.

BRIDGE_DEV_x_FORWARD_DELAY Default: `BRIDGE_DEV_x_FORWARD_DELAY='15'`

This setting is optional and can also be completely omitted.

Only valid if `BRIDGE_DEV_x_STP='yes'` is set!

If one connection of the bridge was deactivated or if a connection is added to the bridge it takes the given time in seconds $\times$ 2 until the connection can send data. This parameter is crucial for the time the bridge needs to recognize a dead connection. The time period is calculated in seconds with this formula:

`BRIDGE_DEV_x_MAX_MESSAGE_AGE`$+(2 \times$ `BRIDGE_DEV_x_FORWARD_DELAY`$)$

In standard values this means: $20 + (2 \times 15) = 50$ seconds. The time to recognize a dead connection can be minimized if `BRIDGE_DEV_x_HELLO` is set to 1 second and `BRIDGE_DEV_x_FORWARD_DELAY` is set to 4 seconds. In addition `BRIDGE_DEV_x_MAX_MESSAGE_AGE` has to set to 4 seconds. This leads to: $4 + (2 \times 4) = 12$ seconds. This is as fast as it can get.

BRIDGE_DEV_x_HELLO Default: `BRIDGE_DEV_x_HELLO='2'`

This setting is optional and can also be completely omitted.

Only valid if `BRIDGE_DEV_x_STP='yes'` is set!

The time mentioned in `BRIDGE_DEV_x_HELLO` is the time in seconds in which the so-called 'Hello-message' is sent by the main bridge. These messages are necessary for STP's automatic configuration.

BRIDGE_DEV_x_MAX_MESSAGE_AGE Default: `BRIDGE_DEV_x_MAX_MESSAGE_AGE='20'`

This setting is optional and can also be completely omitted.

Only valid if `BRIDGE_DEV_x_STP='yes'` is set!

The maximum time period the last 'Hello-message' stays valid. If no new 'Hello-message' is received during this period a new main bridge election will be triggered. This is why this value should **never** be lower than 2 × `BRIDGE_DEV_x_HELLO`.

BRIDGE_DEV_x_DEV_x_PORT_PRIORITY Default: `BRIDGE_DEV_x_DEV_x_PORT_PRIORITY='128'`

This setting is optional and can also be completely omitted.

Only valid if `BRIDGE_DEV_x_STP='yes'` is set!

Only relevant if multiple connections with the same `BRIDGE_DEV_x_DEV_x_PATHCOST` have the same destination. If this is the case the connection with lowest priority will be chosen.

Valid values are '0' to '240' in steps of '16'.

BRIDGE_DEV_x_DEV_x_PATHCOST Default: `BRIDGE_DEV_x_DEV_x_PATHCOST='100'`

This setting is optional and can also be completely omitted.

Only valid if `BRIDGE_DEV_x_STP='yes'` is set!

Indirectly specifies the bandwidth for this connection. The lower the value the higher is the bandwidth and therefore the connection gets a higher priority.

The calculation base proposed is 1000000 kbit/s which leads to the traffic costs listed in table 1.1. Please note to use the actual usable bandwidth in the formula when calculating. As a result this leads to significantly lower values than you would expect, especially on wireless lan.

Note: The current IEEE standard from 2004 uses 32 bit numbers for bandwidth calculation which is not supported on Linux yet.

Bandwidth	Setting of `BRIDGE_DEV_x_DEV_x_PATHCOST`
64 kbit/s	15625
128 kbit/s	7812
256 kbit/s	3906
10 Mbit/s	100
11 Mbit/s	190
54 Mbit/s	33
100 Mbit/s	10
1 Gbit/s	1

Table 1.1: Values for BRIDGE_DEV_x_DEV_x_PATHCOST as a function of bandwidth

1.1.6 Notes

A bridge will forward any type of Ethernet data - thus e.g. a regular DSL modem can be used over WLAN as if it had a WLAN interface. No packets that pass the bridge will be examined for any undesirable activities (ie the fli4l packet filter is not active!). Use only after careful consideration of security risks (ie as a WLAN access point). There is also the possibility to activate EBTables support however.

1.1.7 EBTables - EBTables for fli4l

As of Version 2.1.9 fli4l has rudimental EBTables support. By setting `OPT_EBTABLES='yes'` EBTables support will get activated.

This means that all ebtables kernel modules get loaded and the ebtables program on the fli4l-routers will get available. In contrast to the much simplified netfilter configuration through the different filter lists of fli4l it then is necessary to write an ebtables script of your own. This means you have to write the complete ebtables script yourself.

For background informations about EBTables support please read the EBTables documentation at http://ebtables.sourceforge.net.

There is the possibility of issuing ebtables commands on the router before and after setting up the netfilter (`PF_INPUT_x`, `PF_FORWARD_x` etc). To do so, create the files ebtables.pre und ebtables.post in the directory config/ebtables. Ebtables.pre will get executed before and ebtables.post after configuring the netfilter. Please remember that an error in the ebtables scripts may interrupt the boot process of the fli4l-router!

Before using EBTables you should definitely read the complete documentation. By using EBTables the complete behavior of the router may change! Especially filtering by mac: in PF_FORWARD will not work as before.

Have a look at this page giving a small glimpse about how the the ebtables support works: http://ebtables.sourceforge.net/br_fw_ia/br_fw_ia.html.

1.1.8 ETHTOOL - Settings for Ethernet Network Adapters

By setting `OPT_ETHTOOL='yes'` the ethtool program will be copied to the fli4l router in order to be used by other packages. By the help of this program, various settings of Ethernet network cards and drivers can be displayed and changed.

ETHTOOL_DEV_N Specify the number of settings to set at boot time.

Default: `ETHTOOL_DEV_N='0'`

ETHTOOL_DEV_x `ETHTOOL_DEV_x` indicates for which network device the settings should apply.

Example: `ETHTOOL_DEV_1='eth0'`

ETHTOOL_DEV_x_OPTION_N `ETHTOOL_DEV_x_OPTION_N` indicates the number of settings for the device.

ETHTOOL_DEV_x_OPTION_x_NAME

ETHTOOL_DEV_x_OPTION_x_VALUE The variable `ETHTOOL_DEV_x_OPTION_x_NAME` gives the name and `ETHTOOL_DEV_x_OPTION_x_VALUE` the value of the setting to be changed.

Following is a list of options and possible values activated by now:

- speed 10|100|1000|2500|10000 expandable by HD or FD (default FD = full duplex)
- autoneg on|off
- advertise %x
- wol p|u|m|b|a|g|s|d

Example:

```
OPT_ETHTOOL='yes'
ETHTOOL_DEV_N='2'
ETHTOOL_DEV_1='eth0'
ETHTOOL_DEV_1_OPTION_N='1'
ETHTOOL_DEV_1_OPTION_1_NAME='wol'
ETHTOOL_DEV_1_OPTION_1_VALUE='g'
ETHTOOL_DEV_2='eth1'
ETHTOOL_DEV_2_OPTION_N='2'
ETHTOOL_DEV_2_OPTION_1_NAME='wol'
ETHTOOL_DEV_2_OPTION_1_VALUE='g'
ETHTOOL_DEV_2_OPTION_2_NAME='speed'
ETHTOOL_DEV_2_OPTION_2_VALUE='100hd'
```

Further informations about ethtool can be found here: http://linux.die.net/man/8/ethtool

1.1.9 Example

For understanding a simple example is certainly helpful. In our example we assume 2 parts of a building which are connected by 2 x 100 Mbit/s lines. Four separate networks should be routed from one building to the other.

To achieve this a combination of bonding (joining the two physical lines) VLAN (to transport several separate networks on the bond) and bridging (to link the different nets to the bond/VLAN) is used. This has been tested successful on 2 Intel e100 cards and 1 Adaptec 4-port card ANA6944 in each building's router. The two e100 have the device names `'eth0'` and `'eth1'`. They are used for connecting the building. Intel e100's are the only cards known to work flawlessly with VLAN by now. Gigabit-cards should work too. The 4 ports of the multiport-card are used for the networks and have device names `'eth2'` to `'eth5'`.

At first the two 100 Mbit/s lines will be bonded:

```
OPT_BONDING_DEV='yes'
BONDING_DEV_N='1'
BONDING_DEV_1_DEVNAME='bond0'
BONDING_DEV_1_MODE='balance-rr'
BONDING_DEV_1_DEV_N='2'
BONDING_DEV_1_DEV_1='eth0'
BONDING_DEV_1_DEV_2='eth1'
```

This creates the device `'bond0'`. Now the two VLANs will be built on this bond. We use VLAN-IDs 11, 22, 33 und 44:

```
OPT_VLAN_DEV='yes'
VLAN_DEV_N='4'
VLAN_DEV_1_DEV='bond0'
VLAN_DEV_1_VID='11'
VLAN_DEV_2_DEV='bond0'
VLAN_DEV_2_VID='22'
VLAN_DEV_3_DEV='bond0'
VLAN_DEV_3_VID='33'
VLAN_DEV_4_DEV='bond0'
VLAN_DEV_4_VID='44'
```

Over this two VLAN connections the bridge into the networks segments will be built. Routing is not necessary this way.

```
OPT_BRIDGE_DEV='yes'
BRIDGE_DEV_N='4'
BRIDGE_DEV_1_NAME='_VLAN11_'
BRIDGE_DEV_1_DEVNAME='br11'
BRIDGE_DEV_1_DEV_N='2'
BRIDGE_DEV_1_DEV_1='bond0.11'
BRIDGE_DEV_1_DEV_2='eth2'
BRIDGE_DEV_2_NAME='_VLAN22_'
BRIDGE_DEV_2_DEVNAME='br22'
BRIDGE_DEV_2_DEV_N='2'
BRIDGE_DEV_2_DEV_1='bond0.22'
BRIDGE_DEV_2_DEV_2='eth3'
BRIDGE_DEV_3_NAME='_VLAN33_'
BRIDGE_DEV_3_DEVNAME='br33'
BRIDGE_DEV_3_DEV_N='2'
BRIDGE_DEV_3_DEV_1='bond0.33'
BRIDGE_DEV_3_DEV_2='eth4'
BRIDGE_DEV_4_NAME='_VLAN44_'
BRIDGE_DEV_4_DEVNAME='br44'
BRIDGE_DEV_4_DEV_N='2'
BRIDGE_DEV_4_DEV_1='bond0.44'
BRIDGE_DEV_4_DEV_2='eth5'
```

As a result all 4 Nets are connected with each other absolutely transparent and share the 200 Mbit/s connection. Even with a failure of one 100 Mbit/s line the connection will not fail. If necessary EBTables support can also be activated e.g. to activate certain packet filter.

This configuration is set up on two fli4l routers. I think this is an impressive example what the advanced_networking package can do.

List of Figures

List of Tables

Index