Paquetage ADVANCED_NETWORKING
Version 3.10.18

Frank Meyer
courriel: frank@fli4l.de

L'équipe fli4l
courriel: team@fli4l.de

15 septembre 2019

Table des matières

1 Documentation du paquetage ADVANCED_NETWORKING

1.1 Advanced Networking

Avec le paquetage advanced_networking il est possible d'élargir les fonctionnalités du routeur fli4l en utilisant une liaison VLAN (ou réseau local virtuel), la fonction bridge (ou pont), et la fonction bonding. Il supporte aussi EBTables que l'on peut activer, ainsi il sera possible de mettre un filtre transparent dans les paquets voir (http://ebtables.sourceforge.net/).

En règle générale le paquetage advanced_networking peut s'installer avec tous les autres paquetages :

Ce logiciel a été pensé uniquement pour les utilisateurs qui on une bonne connaissance des réseaux. En particulier il est nécessaire d'avoir des connaissances solides sur le routage.

En activant EBTables vous pouvez rencontrer des problèmes très inhabituelles, si vous ne connaissez pas à 100% les différents impacts des couches 2 et 3. Il faut activer EBTables pour travailler avec certaines règles de filtrages de paquets, qui est totalement différent par rapport à se que l'on a vu.

1.1.1 Relais broadcast - Transmission par IP broadcast

On peut utiliser d'un relais broadcast pour une transmission par IP broadcast via une autre interface. Ce dispositif est nécessaire pour certaines applications, il utilise le broadcast pour la transmission sur le réseau (par exemple avec l'utilitaire QNAP Finder), en généralement le broadcast ne peut pas transmètre sur le réseau à travers un routeur. Si vous utilisez un relais broadcast ce problème peut être contourné.

Un relais broadcast diffuse toujours les paquets broadcast à toutes les interfaces connectées. Cela signifie qu'il n'est pas nécessaire de configurer un autre relais broadcast pour les échangés entre les interfaces. En outre, il n'est pas souhaitable d'inclure plusieurs relais broadcast pour une même interface.

OPT_BCRELAY Transmission par broadcast

Par défaut : `OPT_BCRELAY='no'`

Si vous indiquez `'yes'` dans cette variable le relais broadcast sera activé. Si vous indiquez `'no'` vous déactivez complètement le paquetage du relais broadcast.

BCRELAY_N Par défaut : `BCRELAY_N='0'`

Dans cette variable vous indiquez le nombre de relais broadcast à configurer.

BCRELAY_x_IF_N Par défaut : `BCRELAY_x_IF_N='1'`

Dans cette variable vous indiquez le nombre d'interfaces qui sont affectés au relais broadcast.

BCRELAY_x_IF_x Par défaut : `BCRELAY_x_IF_x="`

Dans cette variable vous indiquez le nom de l'interface qui sera affectés au relais broadcast

Pour plus de précisions voici un exemple avec un ordinateur sur le réseau interne (connecté sur `eth0`) dans lequel un utilitaire (par exemple QNAP Finder) est installé, le NAS est dans un autre réseau (connecté sur `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 - Regroupées plusieurs cartes réseaux pour avoir un seul lien

On entend par bonding le regroupement d'au moins deux cartes réseau, qui peuvent être également de différents types (c'est-à-dire 3Com et Intel) et de vitesse (10 Mbit/s ou 100 Mbit/s) pour avoir une seule connexion. vous pouvez raccorder soit directement à un ordinateur linux, ou soit à un switch. ainsi par ex. du routeur fli4l au switch vous serez connecté à 200 Mbit/s en Full-Duplex sans grande difficulté. Toute personne qui s'intéresse au bonding doivent lire la documentation (bonding.txt) dans le dossier kernel. Les noms des variables bonding utilisées sont dans la mesure du possible similaires. Sous le kernel 2.6.x de Linux le fichier bonding.txt se trouve dans le répertoire Documentation/networking du kernel source.

OPT_BONDING_DEV Par défaut : `OPT_BONDING_DEV='no'`

Avec `'yes'` vous activez le paquetage bonding. Si vous indiquez `'no'` vous désactivez complètement le paquetage bonding.

BONDING_DEV_N Par défaut : `BONDING_DEV_N='0'`

Nombre de périphérique à configurer.

BONDING_DEV_x_DEVNAME Par défaut : `BONDING_DEV_x_DEVNAME="`

Vous devez indiquer ici un nom pour l'unité de liaison. Le nom doit commencer obligatoirement par 'bond' et doit être suivi par un nombre '0'. Les numéros des équipements bondings ne doivent pas commencer par un '0' et se suivre. Voici par ex. quelques noms 'bond0', 'bond8' ou 'bond99'.

BONDING_DEV_x_MODE Par défaut : `BONDING_DEV_x_MODE="`

Il existe plusieurs méthodes de bonding. Le mode par défaut est Round-Robin (l'équilibrage de charge) 'balance-rr'. Les méthodes possibles sont les suivantes :

balance-rr Round-Robin (équilibrage de charge) : les données sont transmises séquentiellement de la première à la dernière interface. Ce mode permet à la fois l'équilibrage de charge et la tolérance de pannes.

active-backup Active Backup (Sauvegarde active) : seul une interface active est réellement utilisé. En cas de panne, l'interface active suivante prend la relève. L'adresse MAC du bond est visible uniquement sur un port (adaptateur réseau), pour ne pas embrouiller le switch. Ce mode permet la tolérance de pannes.

balance-xor XOR-Methode (mode XOR) : une interface est affectée à l'envoi vers une même adresse MAC. Ainci les transferts sont parallélisés et le choix de l'interface suit les règles [(Adresse-MAC-source XOR Adresse-MAC-destination) modulo nombre

d'interfaces]. Ce mode offre à la fois d'équilibrage de charge (ou Load-Balancing) et la tolérance de pannes.

broadcast Mode broadcast (diffusion) : les données sont envoyées à toutes les interfaces actives.

802.3ad IEEE 802.3ad agrégation dynamique des liens : permet de créer des groupes qui partagent le même paramétrage. L'intérêt est de disposer d'un mode qui ne nécessite pas forcément de configuration manuelle, les liens se découvrent mutuellement et sont agrégés automatiquement.

Conditions :

— Les pilotes des interfaces doivent supporter le dispositif ethtool, pour le contrôle de la vitesse et du mode duplex dans chaque périphérique.
— Implique que le switch, gère le mode IEEE 802.3ad pour l'agrégation dynamique des liens.

balance-tlb Équilibrage de charge auto-adaptatif en émission : seule la bande passante en sortie est adapté à la charge de chaque interface active (Load-Balancing), (elle est calculé en fonction de la vitesse). le flux entrant est affecté à l'interface courante. Si celle-ci devient inactive, une autre prend alors l'adresse MAC de l'interface inactive et devient l'interface courante.

Conditions :

— Les pilotes des interfaces doivent supporter le dispositif ethtool, pour le contrôle de vitesse et du mode duplex, cela pour chaque interface

balance-alb Équilibrage de charge auto-adaptatif en émission et en réception : ce mode inclut en plus du mode balance-tlb un Load-Balancing (ou charge équilibrée) sur le flux entrant et seulement pour le trafic IPV4. L'équilibrage est réalisé au niveau des réponses ARP. Le pilote du bonding intercepte les réponses des clients pour y réécrire l'adresse physique (ou adresse MAC) de l'une des interfaces du lien tout en tenant compte des spécificités du protocole ARP. La répartition entre les différentes interfaces, ce fait de façon séquentiel (Round-Robin)

Réception du trafic créé par le serveur et équilibrage les charges. Quand le client envoie une requête ARP, le pilote bonding récupère les informations l'IP du client dans l'ARP. Lorsque la réponse ARP du bonding revient au client, celui-ci récupère l'adresse physique (ou adresse MAC) de l'une des interfaces du pilote bonding. Le problème résulte dans la négociation des requêtes ARP pour l'équilibrage de charge, à chaque fois qu'une requête ARP est diffusée il utilise l'une des adresses physiques du lien donc l'une des deux interfaces. Par conséquent, les clients récupèrent l'adresse physique du lien pour l'équilibrage de charge et reçoivent le trafic descendant de l'interface active. Cela est pris en charge par l'envoi d'une mise à jour (réponse ARP) à l'ensemble des clients, assignant individuellement l'adresse physique de telle sorte à redistribuer le trafic. L'équilibrage de charge est réparti de façon séquentielle (Round-Robin) parmi le groupe d'interfaces pour un plus grand débit dans le bond.

Lorsqu'une connexion est rétablie ou une nouvelle interface est rajoutée sur le bond, le trafic entrant est redistribué entre toutes les interfaces activent du bond, en initiant les réponses ARP de tous les clients en récupérant leur adresse MAC. La valeur updelay (détaillée ci-dessous) doit être réglé sur une valeur supérieure ou égale à la transmission retardée du switch (forwarding delay), de sorte que les réponses ARP les clients ne soient pas bloquées par le switch.

Conditions :

— Les pilotes des interfaces doivent supporter le dispositif ethtool, pour le contrôle de la vitesse et du mode duplex dans chaque périphérique (ou interface).

— Supporte les pilotes de base, avec l'utilisation de l'adresse physique (ou adresse MAC) de interface lorsque celle-ci est active. Cette adresse physique est nécessaire afin qu'il y est toujours qu'une seule interface utilisé dans l'équipement du bond (avec curr_active_slave) (ou interface active) tout en ayant une adresse physique unique pour chaque interface dans le bond. Si le curr_active_slave (ou interface active) échoue son adresse physique est échangé avec la nouvelle curr_active_slave qui a été choisie.

BONDING_DEV_x_DEV_N Par défaut : `BONDING_DEV_x_DEV_N='0'`

On indique ici le nombre d'interface pour le bonding. Si par exemple vous avez pour le bonding 'eth0' et 'eth1' vous indiquez '2' dans la variable (pour les deux interfaces eth).

BONDING_DEV_x_DEV_x Par défaut : `BONDING_DEV_x_DEV_x="`

On indique ici le nom de l'interface qui sera actif dans le bonding pour l'agrégation du lien, vous indiquez par exemple 'eth0' dans la variable. Veuillez noter, l'interface que vous utilisez pour le bonding doit être exclusif au bonding, cet interface ne doit pas être utilisée pour d'autres raccordements tels que le modem DSL, le Bridge, le VLAN ou dans le fichier de configuration base.txt

BONDING_DEV_x_MAC Par défaut : `BONDING_DEV_x_MAC="`

Cette variable est optionnelle et peut être ignoré.

Vous pouvez paramétrer ici l'adresse MAC de l'interface bonding que vous utilisez par défaut, qui sera utilisée pour l'agrégation des liens. Vous n'êtes pas obligé de paramétrer l'adresse MAC de l'interface bonding par défaut.

BONDING_DEV_x_MIIMON Par défaut : `BONDING_DEV_x_MIIMON='100'`

Cette variable est aussi optionnelle et peut également être ignoré.

Indique la fréquence de surveillance (en millisecondes) sur l'état de fonctionnement des interfaces du bonding, avec la valeur '0' la surveillance de MIIMON est désactivée.

BONDING_DEV_x_USE_CARRIER Par défaut : `BONDING_DEV_x_USE_CARRIER='yes'`

Cette variable est aussi optionnelle et peut également être ignoré.

Si la surveillance du statuts est activé avec MIIMON et si la (variable est sur 'yes') on demande alors à l'algorithme de surveillance d'utiliser la primitive netif_carrier_ok() à la place de la surveillance des registres MII ou ETHTOOL ioctl(). Toutes les cartes ne supportent pas la primitive netif_carrier_ok() auquel cas le lien est toujours considéré actif, vous pouvez (désactiver ce système par 'no')

BONDING_DEV_x_UPDELAY Par défaut : `BONDING_DEV_x_UPDELAY='0'`

Cette variable est aussi optionnelle et peut également être ignoré.

La valeur de ce paramètre doit être un multiple de la valeur `BONDING_DEV_x_MIIMON`, permet de spécifier la valeur en millisecondes permettant d'activer une interface lors de la détection d'une reconnexion sur le lien (par ex. l'interface eth).

BONDING_DEV_x_DOWNDELAY Par défaut : `BONDING_DEV_x_DOWNDELAY='0'`

Cette variable est aussi optionnelle et peut également être ignoré.

La valeur de ce paramètre doit être un multiple de la valeur `BONDING_DEV_x_MIIMON`, permet de spécifier la valeur en millisecondes permettant de désactiver une interface après la détection d'un problème (par ex. l'interface eth). De cette façon, la connexion du dispositif bonding est toujours active, jusqu'à ce que l'état de la liaison redevient "active".

BONDING_DEV_x_LACP_RATE Par défaut : `BONDING_DEV_x_LACP_RATE='slow'`

Cette variable est aussi optionnelle et peut également être ignoré.

Si vous avez paramétré dans la variable `BONDING_DEV_x_MODE="` le mode '802.3ad', cette variable permet de paramétrer la fréquence d'envoi des paquets sur (un switch ou sur un ordinateur Linux), dans le cas l'agrégation dynamique 802.3ad. le paramètre 'slow' transmission des paquets toutes les 30 secondes, le paramètre 'fast' transmission des paquets toutes les 1 secondes.

BONDING_DEV_x_PRIMARY Par défaut : `BONDING_DEV_x_PRIMARY="`

Cette variable est aussi optionnelle et peut également être ignoré.

Avec cette variable vous pouvez spécifier un périphérique de sortie primaire si le mode est réglé sur 'active-backup'. Ceci est particulièrement utile lorsque les différentes interfaces ont des vitesses différentes. les interfaces (eth0, eth2, etc) peuvent être utilisées comme interface primaire. Si une valeur est entrée dans cette variable et que l'interface est en ligne, elle sera utilisée comme premier moyen de sortie. Lorsqu'il y a un problème sur cette interface, le trafic est dirigé vers une autre interface de secours. Cependant en cas de réactivation de l'interface prioritaire, le trafic sera de nouveau redirigé vers cette interface. On favorise l'interface prioritaire la plus rapide par ex. 1000 Mbit/s les autre à 100 Mbit/s. Si l'interface à 1000 Mbit/s tombe en panne et si celle-ci est réactivé cela est beaucoup plus avantageux que de rester sur l'interface de secours à 100 Mbit/s.

BONDING_DEV_x_ARP_INTERVAL Par défaut : `BONDING_DEV_x_ARP_INTERVAL='0'`

Cette variable est aussi optionnelle et peut également être ignoré.

Permet de spécifier la fréquence en millisecondes, pour vérifier (avec leur réponse ARP) l'intervalle avec l'adresses IP spécifiées dans `BONDING_DEV_x_ARP_IP_TARGET_x`. Si aucune réponse n'est reçus après cette demande, l'interface est considérée comme perdue. Si la surveillance par ARP est utilisée dans le mode-Load-Balancing (mode 0 ou 2), le switch doit être configuré dans le mode qui permet de distribuer des paquets à travers tous les liens - par ex. Round-Robin. Si le switch est configuré pour distribuer les paquets dans le mode XOR, toutes les réponses des cibles ARP serons reçues sur le même lien et pourrait entraîner l'échec des autres membres du réseau. La valeur par défaut est '0' qui désactive la surveillance par ARP.

BONDING_DEV_x_ARP_IP_TARGET_N Par défaut : `BONDING_DEV_x_ARP_IP_TARGET_N="`

Cette variable est aussi optionnelle et peut également être ignoré.

On indique dans cette variable le nombre d'adresse IP, pour la surveillance ARP. On peut indiquer jusqu'à 16 adresses IP maximum.

BONDING_DEV_x_ARP_IP_TARGET_x Par défaut : `BONDING_DEV_x_ARP_IP_TARGET_x="`

Cette variable est aussi optionnelle et peut également être ignoré.

On indique ici les adresses IP à surveiller, si la variable `BONDING_DEV_x_ARP_INTERVAL` > 0 est supérieur à 0. Les adresses seront spécifiées dans le format ddd.ddd.ddd.ddd, ces adresses seront contrôlées par les requêtes ARP pour établir la qualité de la connexion. Au moins une adresse IP doit être listée pour que la surveillance ARP fonctionne.

1.1.3 VLAN - Supporte le 802.1Q

Le VLAN supporte le standard IEEE 802.1Q, à condition que les connexions sont en association avec un switch approprié. La gestion basée sur un VLAN par ports n'est *pas* approprié. Une introduction générale au sujet du VLAN se trouve à l'adresse suivante http://www.inetdoc.net/articles/inter-vlan-routing/inter-vlan-routing.vlan.html ils est justement approprié pour l'entrée en matière du VLAN. Vous trouverez d'autres information et documentation sur le Net qui concerne ce sujet par ex. http://fr.wikipedia.org/wiki/VLAN

Faites attention S.V.P., toutes les cartes réseaux ne supportent pas le WLAN. Certaines cartes réseaux ne peuvent pas du tout traiter VLANs, d'autres ont besoin d'un MTU adapté et d'autres cartes fonctionnent parfaitement sans problèmes. L'auteur du paquetage advanced_networking utilise des cartes réseaux Intel avec le pilote 'e100' sans aucun problème, le réglage du MTU n'est pas nécessaire. Avec le pilote 3COM '3c59x' il est nécessaire d'adapter le MTU, le MTU doit être réglé sur 1496, autrement la carte ne fonctionnera pas correctement. Le pilote 'starfire' ne fonctionne pas correctement si l'interface VLAN est sur un bridge, dans ce cas, aucun paquet ne peut plus être reçu. pour ceux qui veulent travailler avec VLAN ils doivent veiller à ce que les pilotes de cartes réseaux pour le VLAN soient correctement pris en charge par Linux.

OPT_VLAN_DEV Par défaut : `OPT_VLAN_DEV='no'`

Avec `'yes'` vous activez le programme VLAN, avec `'no'` vous le désactivé.

VLAN_DEV_N Par défaut : `VLAN_DEV_N="`

Nombre d'interface VLAN à configurer.

VLAN_DEV_x_DEV Par défaut : `VLAN_DEV_x_DEV="`

Le nom de l'interface, le switch compatible avec le VLAN. Cela peut être par ex. `'eth0'`, `'br1'` ou `'eth2'`.

VLAN_DEV_x_VID Par défaut : `VLAN_DEV_x_VID="`

Ici on paramètre l'identification de l'interface VLAN, le nom de l'interface VLAN est identifié avec le préfix 'ethX' (le '0' n'est plus le référent de l'interface). Par ex. '42' est le nom de l'interface VLAN, il sera indiqué sur le routeur fli4l `'eth0.42'`.

Les interfaces VLAN sur le routeur fli4l sont toujours appelées '<device>.<vid>'. Donc, si j'ai une interface eth pour mon réseau VLAN, un switch compatible, et si je veut configurer 3 interfaces VLANs 10, 11 et 23 virtuelle avec l'interface eth sur mon routeur fli4l, je paramètre la variable `VLAN_DEV_x_DEV='ethX'` et l'ID du VLAN dans la variable `VLAN_DEV_x_VID="`. Mais, comme toujours, un exemple vaut mieux que mille mots, voici l'exemple :

```
OPT_VLAN_DEV='yes'
VLAN_DEV_N='3'
VLAN_DEV_1_DEV='eth0'
VLAN_DEV_1_VID='10' # Nom de l'interface : eth0.10
VLAN_DEV_2_DEV='eth0'
VLAN_DEV_2_VID='11' # Nom de l'interface : eth0.11
VLAN_DEV_3_DEV='eth0'
VLAN_DEV_3_VID='23' # Nom de l'interface : eth0.23
```

S.V.P., pensez toujours à examiner le MTU de l'interface. L'en-tête de la trame Ethernet est supérieur de 4 octets pour le VLAN. Pour certaines interface et si c'est nécessairement vous devrez changer le MTU et indiquer la valeur 1496

1.1.4 Périphérique MTU - Réglage du MTU

Dans de rare circonstance, il peut être nécessaire de régler le MTU d'une interface. Par exemple 100% des cartes réseau ne sont pas compatibles avec le VLAN et certaines on besoin d'un réglage MTU. N'oubliez pas que seul un petit nombre de cartes réseaux sont en mesure de traiter les trames Ethernet avec plus de 1500 octets !

DEV_MTU_N Par défaut : `DEV_MTU_N="`

Cette variable est optionnelle et peut être ignoré.

Indiquez ici le nombre d'interface pour lequelle vous devez modifier la valeur MTU.

DEV_MTU_x Par défaut : `DEV_MTU_x="`

Cette variable est aussi optionnelle et peut également être ignoré.

Indiquer ici le nom de l'interface suivi du réglage MTU. Ces deux éléments sont séparés par un espace. Par ex. pour `'eth0'` le MTU sera `'1496'`, vous pouvez voir ici l'exemple :

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

1.1.5 BRIDGE - Pont Ethernet pour fli4l

Il s'agit ici d'un bridge Ethernet (ou pont Ethernet) authentique, il travaille selon le protocole Spanning Tree. Le fonctionnement de l'ordinateur avec le bridge semble travailler comme un switch Layer 3 avec la configuration des ports.

Si vous voulez plus d'information sur le bridging, vous pouvez aller voir le site :

La page d'accueil du projet Linux sur le bridge : http://bridge.sourceforge.net/.

Description détaillée du briding standard norme 802.1d à lire obligatoirement : http://standards.ieee.org/getieee802/download/802.1D-2004.pdf. Les informations à partir de la page 153 sont surtout intéressante. Veuillez noter que le code source standard du bonding pour Linux est de 1998. c'est-à-dire qu'il y a seulement 16 bits pour les valeurs du PathCost.

Ici, on peut voir les différentes valeurs du délai d'attente pour le Spanning pour le calcule du protocole : http://www.dista.de/netstpclc.htm

Sur la page STP, différent moyen pour travailler agréable, exemple : http://web.archive.org/web/20060114052801/http://www.zyxel.com/support/supportnote/ves1012/app/stp.htm

OPT_BRIDGE_DEV Par défaut : `OPT_BRIDGE_DEV='no'`

Avec `'yes'` le paquetage bridge est activé, avec `'no'` il est désactivé.

BRIDGE_DEV_BOOTDELAY Par défaut : `BRIDGE_DEV_BOOTDELAY='yes'`

Cette variable est optionnelle et peut être ignoré.

Le bridge a au moins 2 × `BRIDGE_DEV_x_FORWARD_DELAY` de délai attente en seconde, c'est le temps nécessaire pour l'activer, voir si ce laps de temps est nécessaires pour le démarrage des périphériques sur fli4l, par ex. envoyer des messages sur syslog ou se connecter par DSL. Si l'entrée est laissé à `'yes'` il y a automatiquement 2 × `BRIDGE_DEV_x_FORWARD_DELAY` de délai attente. Si le bridge n'est pas directement nécessaire au démarrage, vous pouvez indiquer le paramètre `'no'` pour accélérer le processus de démarrage du routeur fli4l.

BRIDGE_DEV_N Par défaut : `BRIDGE_DEV_N='1'`

Vous indiquez ici le nombre de bridge, ils sont indépendants les uns des autres. Chaque bridge est à considérer différents, il sont complètement isolé. C'est valable en particulier pour le réglage de la variable `BRIDGE_DEV_x_STP`. Le bridge devient un périphérique virtuel avec son nom `'br<numéro>'`.

BRIDGE_DEV_x_NAME Par défaut : `BRIDGE_DEV_x_NAME="`

Le nom du bridge est symbolique. Ce nom peut être utilisé par d'autres paquetages qui fonctionne avec le bridge indépendamment des noms des l'interfaces.

BRIDGE_DEV_x_DEVNAME Par défaut : `BRIDGE_DEV_x_DEVNAME="`

Chaque bridge nécessite un nom sous la forme de `'br<numéro>'`. Le <numéro> est un nombre compris entre '0' et '99'. Les entrées possibles sont `'br0'`, `'br9'` ou `'br42'`. Les noms peuvent être choisis indifféremment, les premiers bridge peuvent s'appeler `'br3'` et le deuxième `'br0'`.

BRIDGE_DEV_x_DEV_N Par défaut : `BRIDGE_DEV_x_DEV_N='0'`

Indiquer ici le nombre de périphérique qui doivent être attaché aux bridge, combien d'interface réseau peut comporter le bridge ? Il peut en avoir `'0'`, si on utilise le bridge seulement comme un espace réservé pour une adresse IP qui sera alors repris par un tunnel VPN attaché aux bridge.

BRIDGE_DEV_x_DEV_x_DEV On indique ici les interfaces réseaux attachées au bridge. Peut être enregistré une interface eth (par ex. `'eth0'`), un bonding (par ex. `'bond0'`) ou également un périphérique Vlan (par ex. `'vlan11'`). Un périphérique intégré ici ne peut plus être utilisé ailleurs et ne peut recevoir aucune adresse IP.

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

BRIDGE_DEV_x_AGING Par défaut : `BRIDGE_DEV_x_AGING='300'`

Cette variable est optionnelle et peut être ignoré.

On indique ici le temps pour supprimer les anciennes entrée MAC dans la table du bridge. Si pendant temps spécifié ici en secondes, l'ordinateur n'a pas envoyé ou reçue de données par la carte réseau les adresses MAC la table MAC du Bridge sera supprimé.

BRIDGE_DEV_x_GARBAGE_COLLECTION_INTERVAL Par défaut : `BRIDGE_DEV_x_GARBAGE_COLLECTI`

Cette variable est aussi optionnelle et peut également être ignoré.

On indique ici le temps en seconde avant de faire un «nettoyage». Dans ce cas, les entrées dynamiques du bridge sont vérifiées pour connaître les entrées obsolètes ou invalides. Cela signifie que les anciens liens ne sont plus valides et seront supprimés.

BRIDGE_DEV_x_STP Par défaut : `BRIDGE_DEV_x_STP='no'`

Cette variable est aussi optionnelle et peut également être ignoré.

Le Spanning Tree Protocole permet d'entretenir de multiples liens avec d'autres switchs. De cette manière la redondance garantit l'état de marche du réseau en cas de panne. Sans la mise en place du STP, la redondance entre les liens ne serai pas possible, le réseau ne pourrais pas fonctionner. STP essaye toujours d'utiliser la connexion plus rapide entre deux switchs, ainsi l'installation de deux circuits différentes est judicieuse. On pourrait par ex. installer pour une connexion 1000 Mbit/s en tant que lien principal et utiliser une deuxième connexion à 100 Mbit/s en sécurité.

Vous pouvez consulter cette page, il y a un bon article avec quelques informations de fond : http://fr.wikipedia.org/wiki/Spanning_tree_protocol.

BRIDGE_DEV_x_PRIORITY Par défaut : `BRIDGE_DEV_x_PRIORITY="`

Cette variable est aussi optionnelle et peut également être ignoré.

seulement valable si la variable `BRIDGE_DEV_x_STP='yes'` est paramétré sur yes !

Quelle priorité pour le bridge ? Le bridge avec la plus petite priorité dans l'architecture bridge sera élu par rapport au bridge principal. Chaque bridge devrait avoir une priorité différente. Veuillez considérer que le bridge avec la plus faible priorité devrait disposer de la plus grande bande passante. Puisque ces toutes les 2 secondes (à modifier dans `BRIDGE_DEV_x_HELLO`) que les paquets sont envoyé, également le flux de données restant est acheminé sur celui-ci.

Les valeurs valides sont de `'0'` à `'61440'` par multiples de 4096.

BRIDGE_DEV_x_FORWARD_DELAY Par défaut : `BRIDGE_DEV_x_FORWARD_DELAY='15'`

Cette variable est aussi optionnelle et peut également être ignoré.

Seulement valable si la variable `BRIDGE_DEV_x_STP='yes'` est paramétré sur yes !

Si un lien du bridge est désactivé et qui doit être réactivé, ou bien un lien nouvellement ajoutés sur le bridge, on indique dans cette variable un laps de temps (en secondes) $\times$ 2 avant que les données soit transmis de nouveau sur le lien. Ce paramètre est décisif pour reconnaître la durée nécessaire d'une connexion morte sur le bridge. Ce laps de temps est calculé en secondes avec la formule suivante :

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

Il en découle avec les valeurs par défaut : $20 + (2 \times 15) = 50$ Seconde. la durée nécessaire pour reconnaître d'une connexion morte il peut être réduit, si la variable `BRIDGE_DEV_x_HELLO` et à 1 seconde et si la variable `BRIDGE_DEV_x_FORWARD_DELAY` est à 4 secondes, la variable `BRIDGE_DEV_x_MAX_MESSAGE_AGE` doit être réglée à 4 secondes. Le calcul de la valeur sera la suivante : $4 + (2 \times 4) = 12$ Secondes. Plus rapide cela ne fonctionne pas.

BRIDGE_DEV_x_HELLO Par défaut : `BRIDGE_DEV_x_HELLO='2'`

Cette variable est aussi optionnelle et peut également être ignoré.

Seulement valable si la variable `BRIDGE_DEV_x_STP='yes'` est paramétré sur yes !

Avec la variable `BRIDGE_DEV_x_HELLO` on donne l'intervalle temps en secondes dans le quelle les Hello à vrai dire les messages sont envoyés sur le bridge principal. Ces messages sont nécessaires à la configuration automatique de STP.

BRIDGE_DEV_x_MAX_MESSAGE_AGE Par défaut : `BRIDGE_DEV_x_MAX_MESSAGE_AGE='20'`

Cette variable est aussi optionnelle et peut également être ignoré.

Seulement valable si la variable `BRIDGE_DEV_x_STP='yes'` est paramétré sur yes !

On indique ici la durée de validité maximum du dernier Hello ou message. Si pendant ce temps (en secondes) aucun Hello ou message est récupéré par le bridge principale, un nouveau bridge principal sera choisi. Par conséquent, cette valeur ne peut être **jamais** plus petit que 2 $\times$ `BRIDGE_DEV_x_HELLO`.

BRIDGE_DEV_x_DEV_x_PORT_PRIORITY Par défaut : `BRIDGE_DEV_x_DEV_x_PORT_PRIORITY='128'`

Cette variable est aussi optionnelle et peut également être ignoré.

Seulement valable si la variable `BRIDGE_DEV_x_STP='yes'` est paramétré sur yes !

Cette variable est important seulement lorsque plusieurs connexions a le même paramètre dans `BRIDGE_DEV_x_DEV_x_PATHCOST` et la même destination. Si tel est le cas, la connexion avec la priorité la plus faible sera retenue.

les valeurs valides sont de '0' à '240' avec un multiple de '16'.

BRIDGE_DEV_x_DEV_x_PATHCOST Par défaut : `BRIDGE_DEV_x_DEV_x_PATHCOST='100'`

Cette variable est aussi optionnelle et peut également être ignoré.

Seulement valable si la variable `BRIDGE_DEV_x_STP='yes'` est paramétré sur yes !

Détermine indirectement la bande passante pour cette connexion. En fonction de la valeur faible ou élevée, plus la bande passante est élevé et plus la connexion est prioritaire.

La base de calcul proposée est 1000000 / kbit/s voici les valeurs listées dans le tableau 1.1. S'il vous plaît, faites attention qu'au calcul la bande passante réelle utilisable qui doit être employée dans la formule. Il en résultera, surtout des valeurs pour le WLAN nettement plus faibles par rapport à ce que l'on pourrait s'y attendre.

Remarque : le standard IEEE actuel de 2004 utilise pour le calcul de bande passante 32 Bit entiers qui ne sont pas encore supporté par Linux.

Bande passante	Valeur de `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 – Les valeurs de la variable BRIDGE_DEV_x_DEV_x_PATHCOST sont en fonction de la bande passante

1.1.6 Remarque

Un bridge transmet tout type de données Ethernet - Par exemple il peut être normal avec un modem DSL via WLAN (ou sans fil) réagir comme une interface sans fil. Ce qui permet, par exemple, d'utiliser uniquement le bridge comme un point d'accès WLAN, il faut être attentif aux risques en matière de sécurité, un contrôle est recommandé. Il ne s'agit pas d'un paquet indésirable qui arrive sur le bridge vienne infecter le réseau (cela veut dire que le filtrage de paquets du bridge n'est pas actif dans fli4l !). Il est possible d'activer EBTables qui est supporté par fli4l pour ce filtrage.

1.1.7 EBTables - EBTables pour fli4l

A partir de la version 2.1.9 fli4l support un EBtables rudimentaire. En mettant la variable `OPT_EBTABLES='yes'` sur yes vous acivez EBTables. Tous les modules EBtables sont chargés dans le Kernel (ou noyau) et le programme ebtables est mis à disposition sur le routeur fli4l. Cela signifie que nous devons écrire le script EBTables complétement soi-même.

Pour plus d'information sur le support EBTables vous pouvez lisez la documentation sur le site Web EBTables : http://ebtables.sourceforge.net.

Il y a la possibilité d'indiquer les commandes EBTables avant et après netfilters (comme ceci `PF_INPUT_x`, `PF_FORWARD_x` etc) sur le routeur fli4l. Mettez selon vos besoins, dans le répertoire config/ebtables les fichiers ebtables.pre et ebtables.post. Le fichier ebtables.pre est

exécuté avant la configuration de netfilters, puis le fichier ebtables.post sera exécuté. S'il vous plaît rappelez-vous qu'une erreur dans le scripts ebtables peuvent interrompt le processus de démarrage du routeur fli4l !

Avant d'utiliser le support EBTables vous devez lire complètement la documentation EBTables. Avec utilisation EBTables vous pouvez changer le comportement du routeur fli4l ! Par exemple le filtrage du module Mac : PF_FORWARD ne fonctionnera pas comme d'habitude.

D'une façon très intéressant, sur l'adresse Web qui suit un petit aperçu du fonctionnement du support EBTables : http://ebtables.sourceforge.net/br_fw_ia/br_fw_ia.html.

1.1.8 ETHTOOL - Paramètres pour carte réseau Ethernet

Si vous activez cette variable `OPT_ETHTOOL='yes'` le programme ethtool sera copié dans fli4l, d'autres paquetages installé pourront aussi l'utiliser. Grâce à ce programme, différents paramètres de cartes réseaux Ethernet et de pilotes pourront être affichés et modifiés.

ETHTOOL_DEV_N Vous pouvez indiquer ici le nombre de paramètres, qui peuvent être définis au moment du démarrage.

Par défaut : `ETHTOOL_DEV_N='0'`

ETHTOOL_DEV_x Vous spécifiez dans cette variable `ETHTOOL_DEV_x` le périphérique réseau pour lequel les réglages devrait s'appliquer.

Exemple : `ETHTOOL_DEV_1='eth0'`

ETHTOOL_DEV_x_OPTION_N Vous indiquez ici `ETHTOOL_DEV_x_OPTION_N` le nombre de paramètres pour le périphérique.

ETHTOOL_DEV_x_OPTION_x_NAME

ETHTOOL_DEV_x_OPTION_x_VALUE Dans cette variable `ETHTOOL_DEV_x_OPTION_x_NAME` vous donnez un nom et dans celle-ci `ETHTOOL_DEV_x_OPTION_x_VALUE` vous indiquez la valeur à modifier.

Voici une liste d'options et de valeurs possibles, qui pourront être activées :

- speed 10|100|1000|2500|10000 vous pouvez ajouter HD ou FD (par défaut FD = Full-Duplex)
- autoneg on|off
- advertise %x
- wol p|u|m|b|a|g|s|d

Exemple :

```
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'
```

Pour plus d'informations, vous pouvez consulter la documentation sur ethtool : http://linux.die.net/man/8/ethtool

1.1.9 Exemples

Un exemple simple est certainement plus utile pour comprendre. Nous allons dans notre exemple connecter 2 immeubles, avec un liaison à 2 x 100 Mbit/s. A ce sujet il y aura d'un bâtiment à l'autre 4 réseaux séparés.

Pour réaliser cela, une combinaison bonding (avec 2 connexions à 100 Mbit/s pour une transmission de performance), un VLAN (une agrégation de la ligne pour pouvoir transporter plusieurs réseaux distincts) et un bridging (pour pouvoir installer les réseaux dans le bâtiment avec l'entité Bond/VLAN (tester avec succès avec les cartes 2x Intel e100 et la carte ANA6944 1x Adaptec 4-Port). Dans cet exemple les deux interfaces e100 ce nomme `'eth0'` et `'eth1'`, sont utilisés pour la connexion des bâtiments. Nous n'avons pas d'autres types de carte connus à par Intel e100 qui s'associent sans problème avec le VLAN. en principe les cartes gigabit doivent aussi fonctionner. Les 4 ports de la carte multi-port est utilisé pour les réseaux respectifs ont les noms d'interfaces de `'eth2'` à `'eth5'`.

D'abord on construit les deux lignes à 100 Mbit/s pour le Bonding :

```
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'
```

Nous avons produit le périphérique `'bond0'`. Nous allons construire maintenant les VLANs sur la liaison bonding, nous utiliserons dans cette exemple les ID-VLANs (ou identifiant-VLAN) 11, 22, 33 et 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'
```

Et maintenant sur ces liaisons VLAN, nous allons construire le bridge pour mettre en relation les différent segments du réseau. Un routage n'est pas nécessaire.

```
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'
```

Maintenant, tous les 4 réseaux sont interconnectés et complètement transparent pour partager la connexion à 200 Mbit/s. Même avec une perte de connexion on pourra encore être connecté à 100 Mbit/s. Si nécessaire, vous pouvez activer même l'assistance EBTables, par exemple pour le filtrage de paquets spécifiques.

Cette configuration est construit sur deux routeurs fli4l. Je pense que ces exemples ont permis de montrer les possibilités impressionnantes du paquetage advanced_networking.

Table des figures

Liste des tableaux

Index