Suite de mes aventures avec le routeur Turris Omnia

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 19 mars 2017

https://www.bortzmeyer.org/turris-bis.html

Dans un précédent article <https://www.bortzmeyer.org/turris.html>, j'ai parlé du routeur Turris Omnia <https://omnia.turris.cz/> et de ce qu'on peut faire avec. Ce deuxième article est un assortiment de diverses choses que j'ai faites depuis avec ce routeur.

Il est connecté à Free en ADSL. Plus exactement, l'ADSL arrive sur une Freebox Révolution, configurée en "*bridge*", à laquelle est relié le Turris Omnia, qui est le vrai routeur. À l'origine, j'avais laissé le Freebox Player connecté au Freebox Server, ce qui faisait que la télévision classique et le téléphone marchaient comme avant. Mais comme je voulais regarder les chaînes de télévision depuis un PC, avec VLC et le protocole RTSP (RFC 7826 [1]), il fallait connecter le Freebox Player au routeur. Je me suis beaucoup inspiré de cet excellent article <https://guillaume.vaillant.me/?p=256>. Donc, ce qu'il fallait faire :

— Connecter le câble Ethernet entre le Turris et le Freebox Player (celui-ci n'a donc plus de câble vers le Freebox Server),
— Configurer le commutateur du Turris pour utiliser le VLAN 100, celui sur lequel Freebox Player et Server communiquent.

Attention en jouant avec la configuration du commutateur interne du Turris : une erreur et on se retrouve vite avec des tempêtes de diffusion, qui peuvent aller jusqu'à rendre le routeur inaccessible en Ethernet <https://forum.turris.cz/t/network-storm-bricked-router/2902/2>. J'ai aussi eu un cas amusant où la plupart des paquets étaient bien transmis, sauf ceux de diffusion, ce qui cassait des protocoles comme ARP ou DHCP. Deux conseils : vérifier que le Wi-Fi fonctionne, il peut servir de mécanisme de secours pour se connecter au Turris, si l'Ethernet devient inutilisable. Et bien relire sa configuration avant de la confirmer. Dans le pire des cas, il faudra perdre toute la configuration en remettant le routeur aux réglages d'usine <https://www.turris.cz/doc/en/howto/omnia_factory_reset> (pensez à garder cette documentation **avant** de vous couper votre accès Internet !)

Vu du côté Unix, le Turris a plein d'interfaces réseau. `eth0` rassemble la plupart des ports physiques du commutateur, `eth2` étant le CPU (et le port 4 du commutateur, voir cette discussion sur le forum <https://forum.turris.cz/t/is-the-lan-port-4-special/1287/>). Voici d'ailleurs un schéma :

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc7826.txt

Ensuite, des interfaces virtuelles regroupent plusieurs de ces interfaces (sour l'interface LuCI, vous trouver cette configuration en "*Network -¿ Interfaces*", `https://ROUTER/cgi-bin/luci/admin/network/netwo`
Par exemple, `br-lan` regroupe typiquement `eth0` et `eth2`. Et c'est ainsi que les deux groupes communiquent (sur LuCI, "*Network -¿ Interfaces*" puis "*Edit*" puis "*Physical settings*").

Et les VLAN ? Ils se configurent/s'affichent avec LuCI en "*Network -¿ Switch*", `https://ROUTER/cgi-bin/luci`
Par défaut, tous les ports sont non marqués ("*untagged*") ce qui veut dire que le commutateur ne fait pas de VLAN. Si on branche le Freebox Player sur un port où on active le marquage ("*tagged*" pour le VLAN 100, celui utilisé par les boitiers Freebox) et le Freebox Server sur un autre port marqué 100, les deux boitiers peuvent communiquer, la télévision marche mais, dans ce cas, le réseau local, toujours non marqué, ne peut plus communiquer avec ces boitiers et on n'a donc pas d'accès Internet. La configuration qui marche est donc celle-ci : Le port marqué CPU dans LuCI est celui qui est marqué WAN sur le boitier (je sais, c'est bizarre).

Vous n'aimez pas les copies d'écran, vous préférez des fichiers de configuration ? Pas de problème, cela se configure dans `/etc/config/network` (je n'ai montré que les paramètres pertinents) :

```
config interface 'lan'
option type 'bridge'
option ifname 'eth0 eth2'

config interface 'Freebox'
option type 'bridge'
option proto 'static'
option ifname 'eth0.100 eth1.100'

config switch_vlan
option device 'switch0'
option vlan '1'
option vid '1'
option ports '1 2 3 4 5'

config switch_vlan
option device 'switch0'
option vlan '2'
option ports '0t 5t'
option vid '100'
```

Pour résumer cette configuration : on a deux VLANs, 1 et 100. 100 (deuxième directive `config switch_vlan`) couvre le port 0 (qui est marqué, et où est connecté le Freebox Player) et le port CPU/WAN/5 - connecté au Freebox Server - qui est le seul à être sur deux VLAN (1 en non marqué et 100 en marqué). Le `t` dans la liste des ports indique un marquage ("*tagging*"). L'autre VLAN, 1 (première directive `config switch_vlan`), couvre les autres ports. Pour que les interfaces physiques communiquent, on a deux ponts, `br-lan` (directive `config interface 'lan'`) et `br-Freebox`, qui fait communiquer les deux ports du VLAN 100 (qui arrivent sur des commutateurs différents, regardez le schéma plus haut). Les ports marqués correspondent aux interfaces comportant le numéro du VLAN (comme `eth0.100`, les paquets du VLAN 100 arrivant sur `eth0`)

(Au passage, si vous utilisez LuCI pour configurer, vous devrez cliquer sur "*Save and apply*" pour appliquer votre configuration. Rappelez-vous de bien la vérifier avant. Si vous avez au contraire édité le fichier de configuration à la main, ce sera un `/etc/init.d/network restart`, avec les mêmes précautions.)

Avec tout ça, tout le monde communique, la télé marche (si le Freebox Player affiche au démarrage qu'il ne peut pas communiquer avec le Freebox Server, c'est que vous avez un problème), l'Internet fonctionne, etc. Mais on ne peut toujours pas regarder la télévision avec VLC (`vlc http://mafreebox.freebox.fr/f`

affiche "*live555 demux error : no data received in 10s, aborting*"). La raison en est que RTSP est un protocole un peu spécial (il n'est pas vraiment client/serveur) : certes, le PC se connecte à la Freebox mais le flux vidéo lui-même n'est pas envoyé dans cette connexion, mais séparement sous forme de paquets UDP. Le Turris n'a apparemment pas de mécanisme de suivi des sessions RTSP ("*conntracker*", comme ce module <https://github.com/maru-sama/rtsp-linux>) qui permettrait de transmettre automatiquement ces paquets UDP à la bonne machine. J'ai donc choisi, en suivant cette excellente documentation <http://asdrad.free.fr/neuf/multiposte.html>, de configurer le Turris pour chaque machine. Sur chaque PC du réseau local qui veut regarder des conneries à la télé, il faut fixer le port dans VLC Paramètres -¿ Préférences -¿ Input/Codecs-¿Demuxers -¿ RTP/RTSP. Là on coche la case Options avancées. On voit s'afficher un champ Client port, avec la valeur -1, ce qui signfie que VLC choisit aléatoirement le port d'entrée. On met la valeur de son choix (attention, elle doit être **paire**), par exemple 31336. Il faut aussi configurer le Turris pour transmettre ce port à la bonne machine. (Oui, tout serait plus pratique si `mafreebox.free.fr` avait une adresse IPv6). Dans LuCI, c'est dans "*Network -¿ Firewall*" puis "*Port forwards*" `ROUTER/cgi-bin/luci/admin/network/firewall/forwards` :

Et si vous préférez cette configuration en mode texte, c'est dans `/etc/config/firewall` :

```
config redirect
option target 'DNAT'
option name 'RTSP machine1'
option proto 'udp'
option src 'wan'
option src_dport '31336'
option dest 'lan'
option dest_ip '192.168.X.Y'
option dest_port '31336'
option src_ip '212.27.38.253'
```

Une fois qu'on a ses VLAN comme on veut, on peut s'avachir devant la télé qu'on reçoit sur son PC, ou bien on peut passer à une autre tâche. Installer un disque dur supplémentaire dans l'Omnia et créer des machines virtuelles (les deux tâches sont liées, pour des raisons expliquées plus loin).

Pourquoi un disque supplémentaire, pourquoi ne pas se contenter de la Flash présente ? Cet espace de stockage est largement suffisant (8 Go) pour faire tourner les fonctions de base du routeur, mais il ne l'est plus si on veut installer des applications, par exemple de supervision ou de statistiques, qui vont stocker des données sur le long terme, ou bien si on veut mettre son serveur de messagerie sur le Turris. Ou encore si on veut s'en servir comme NAS. Si on veut réaliser la promesse de la page Web officielle <https://omnia.turris.cz/>, « "*More than just a router. The open-source [sic] center of your home*" », il faudra plus de huit gigas.

D'autant plus que la Flash a un autre problème, elle s'use vite quand on écrit souvent <https://forum.turris.cz/t/why-is-var-on-tmpfs/1327>. Voilà pourquoi, dans OpenWrt, par défaut, `/var` est en mémoire, et donc un équivalent de `/tmp`, qui ne survit pas aux redémarrages. Autrement, des services comme syslog démoliraient la Flash trop vite.

Donc, installons un disque supplémentaire. L'Omnia a un emplacement libre, au dessus de l'emplacement pour carte SIM, où on peut mettre un disque SSD via une interface mSATA. J'ai acheté un Kingston mS200 de 120 Go à 50 € TTC. Mais c'est ensuite que les ennuis commencent. L'emplacement libre dans l'Omnia n'est pas celui qui a le port combiné miniPCIexpress/mSATA, le bon emplacement est occupé par une des deux cartes Wi-Fi, il va donc falloir ouvrir le routeur, et déplacer la carte Wi-Fi. (On pourrait évidemment utiliser un disque externe, connecté en USB mais une de mes motivations pour tout mettre sur le Turris Omnia était de diminuer le nombre de boitiers et de prises de courant.)

La procédure nécessite donc tournevis et une certaine habileté manuelle. Elle est très bien expliquée dans ce film <https://www.youtube.com/watch?v=71_M2N3ga7s> (les commentaires de la vidéo valent également d'être lus). Notez toutefois que dans mon cas, cela n'a pas suffi : les vis du dessus des cartes Wi-Fi ne se défont pas et j'ai donc dû démonter la carte Wi-Fi en l'attaquant de l'autre côté de la carte mère. (Vous trouverez aussi sur le forum Turris des discussions sur cette procédure, comme ici <https://forum.turris.cz/t/which-mpcie-slot-for-msata-solved/964>.) Voici le Turris Omnia ouvert avant l'opération :

Et le même après, la carte Wi-Fi qui était tout à droite ayant été déplacée tout à gauche :

Vous pouvez aussi télécharger une image en haute définition </images/turris-inside-with-msata.jpg>. Attention notamment aux fils qui vont des cartes Wi-Fi aux antennes, ils se défont facilement.

(Notez que, depuis mon bricolage, la carte-mère de l'Omnia a un peu changé <https://framapiaf.org/@Liandri/99514105327922785>.)

Une fois le disque branché et bien branché, la carte mère replacée et le capot fermé, on redémarre le routeur (en priant, si on est croyant). On doit voir un disque en `/dev/sda` (tapez `dmesg | grep sda` après le démarrage). On le formate comme indiqué, par exemple, dans la documentation d'OpenWrt <https://wiki.openwrt.org/doc/howto/storage>. Chez moi, cela donne :

```
# fdisk -l /dev/sda

Disk /dev/sda: 111.8 GiB, 120034123776 bytes, 234441648 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 0263F4A2-3C22-4043-B2ED-32B962569924

Device         Start       End   Sectors  Size Type
/dev/sda1       2048 121636863 121634816   58G Linux filesystem
/dev/sda2  121636864 234441614 112804751 53.8G Linux filesystem

# blkid
/dev/mmcblk0p1: UUID="0eec9a72-3c0f-4222-ab9e-2147243a7c1e" UUID_SUB="6b8deab1-dff4-48fc-a522-395f67d59de8"
/dev/sda1: UUID="cb35ae3d-78f8-49f9-bbbb-efbab97b4a81" TYPE="ext4" PARTUUID="ab197dd0-71d2-446c-80e6-bf8810
/dev/sda2: UUID="df1c2ed7-5728-4629-9d72-263bbf2b5939" TYPE="ext4" PARTUUID="3673e386-6636-40e8-bf08-b32899
/dev/mmcblk0: PTUUID="2cbb06e2" PTTYPE="dos"
```

On peut ensuite monter le disque de la manière OpenWrt habituelle. Voici mon `/etc/config/fstab` :

```
config mount
option enabled '1'
option target '/srv'
option uuid 'cb35ae3d-78f8-49f9-bbbb-efbab97b4a81'

config mount
option enabled '1'
option uuid 'df1c2ed7-5728-4629-9d72-263bbf2b5939'
option target '/var'
```

Pour compléter, notez que ce déplacement d'une carte Wi-Fi va nécessiter de reconfigurer le service Wi-Fi (dans LuCI, "*Network -¿ Wireless*"), la carte passant de `radio1` à `radio2`.

Une fois qu'on a son disque, on peut installer ses machines virtuelles ou plus exactements ses containers. Pourquoi ces machines supplémentaires alors qu'on a déjà un Unix qui tourne parfaitement sur le matériel ? Mon problème était surtout que le nombre de paquetages est très limité sur l'Omnia (cf. la liste <https://api.turris.cz/openwrt-repo/omnia/packages/>). Il n'y a ainsi pas emacs. Les outils de développement sont absents (on peut éventuellement faire de la compilation croisée <https://forum.turris.cz/t/cross-compile-howto/1090>) et, de toute façon, il y a deux bonnes raisons pour ne pas installer plein de choses sur l'Unix OpenWrt de l'Omnia :

— Seuls les paquetages officiels bénéficients d'une fonction essentielle de l'Omnia (notamment pour la sécurité), la mise à jour automatique.
— Le routeur doit router dans tous les cas, et doit donc avoir un jeu de logiciels minimum. Tout ce qu'on rajoute peut créer des problèmes.

Donc, la méthode propre sur Omnia, si on veut des logiciels comme Icinga (pour la supervision) ou des petits utilitaires sympa comme uptimed <https://github.com/rpodgorny/uptimed> ou comme check-soa <https://framagit.org/bortzmeyer/check-soa> (indispensable quand on joue souvent avec le DNS), la méthode propre, donc, est d'installer des machines virtuelles sur l'Omnia.

En fait, ce ne sont pas des machines virtuelles complètes, juste des containers, avec la technique LXC. Contrairement à des vraies machines virtuelles, ils ne fournissent pas une étanchéité complète. Tous utilisent le même noyau, qui ne s'exécute qu'une fois. (C'est d'ailleurs pour cela qu'uptimed <https://github.com/rpodgorny/uptimed> dans un container marche bien : il enregistre l'"*uptime*" du routeur, pas celui du container.) Les containers n'ont pas non plus d'horloge propre et c'est pour cela qu'ils n'ont pas besoin de NTP, celui du routeur suffit.

Autre conséquence du modèle du container, les « machines » doivent tourner avec Linux, pas de FreeBSD sur le Turris Omnia. LXC sur cette machine est bien documenté <https://www.turris.cz/doc/en/howto/lxc>. Voici le processus de création d'un container, avec le choix des systèmes d'exploitation :

```
#  lxc-create -t download -n gandalf
Setting up the GPG keyring
Downloading the image index

---
DIST RELEASE ARCH VARIANT BUILD
---
Turris_OS stable armv7l default 2016-11-27
Turris_OS stable ppc default 2016-11-27
Alpine 3.4 armv7l default 2016-11-27
Debian Jessie armv7l default 2016-11-27
Gentoo stable armv7l default 2016-11-27
openSUSE 13.2 armv7l default 2016-11-27
openSUSE Tumbleweed armv7l default 2016-11-27
Ubuntu Xenial armv7l default 2016-11-27
Ubuntu Yakkety armv7l default 2016-11-27
---

Distribution: Debian
Release: Jessie
Architecture: armv7l

Downloading the image index
Downloading the rootfs
Downloading the metadata
The image cache is now ready
Unpacking the rootfs

---
Distribution Debian version Jessie was just installed into your container.

Content of the tarballs is provided by third party, thus there is no warranty of any kind.
```

Pas d'Arch Linux, je le regrette, donc j'ai mis Debian.

Ensuite, on démarre le container :

```
# lxc-start -n gandalf
```

On s'y attache :

```
# lxc-attach -n gandlf
```

Et on peut configurer le mot de passe, le réseau (je n'ai pas réussi à faire marcher le client DHCP sur les containers, j'ai tout configuré en statique), SSH...(Notez qu'on peut aussi faire tout cela depuis LuCI, "*Services -¿ LXC containers*".) La configuration du container `gandalf` se retrouve dans `/srv/lxc/gandalf/config`. On peut notamment configurer l'adresse MAC du container (attention, si ce n'est pas fait, le container change d'adresse MAC à chaque démarrage, ce qui est excellent pour la vie privée mais moins pour l'administration système, avec arpwatch et NDPMon qui voient une nouvelle machine à chaque fois) :

```
# cat /srv/lxc/gandalf/config
...
# Network configuration
lxc.network.type = veth
lxc.network.link = br-lan
lxc.network.flags = up
lxc.network.name = eth0
lxc.network.script.up = /usr/share/lxc/hooks/tx-off
lxc.network.hwaddr = 21:ae:a4:79:73:16
```

Une fois qu'on a un beau container qui tourne, on peut y installer ses logiciels favoris, comme Icinga (qui, avant, tournait chez moi sur un Raspberry Pi).

Le troisième grand dossier, après les VLAN et l'ajout du disque, c'était la configuration du résolveur DNS. Le Turris utilise par défaut kresd, alias Knot resolver <`https://www.knot-resolver.cz/`>. Intéressant logiciel, quoique ayant encore quelques défauts de jeunesse. Knot marche bien par défaut, et fournit notamment la validation DNSSEC :

```
% dig A www.afnic.fr
...
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 47317
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 6, AUTHORITY: 0, ADDITIONAL: 1
...
;; ANSWER SECTION:
www.afnic.fr. 133 IN CNAME www.nic.fr.
www.nic.fr. 133 IN CNAME lb01-1.nic.fr.
lb01-1.nic.fr. 133 IN A 192.134.5.24
```

Le AD ("*Authentic Data*" dans les "*flags*") indique que le nom est signé et vérifié. Avec un nom pas signé, on n'a pas ce AD :

```
% dig A www.ssi.gouv.fr
...
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 16026
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
...
;; ANSWER SECTION:
www.ssi.gouv.fr. 14400 IN A 213.56.166.109
...
```

Et, si le nom est signé mais erroné, on récupère un `SERVFAIL` (”*Server Failure*”) :

```
% dig A tsc.gov
...
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 28366
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0
...

% dig +cd A tsc.gov
...
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25881
;; flags: qr rd ra cd; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1
...
;; ANSWER SECTION:
tsc.gov. 60 IN A 153.31.112.95
```

(Le `+cd` voulant dire ”*Checking Disabled*”, pour être sûr que le problème vient de DNSSEC.)

Le premier changement que j'ai fait à sa configuration était de couper la transmission (”*forwarding*”) aux résolveurs du FAI (qui sont des résolveurs menteurs <https://www.bortzmeyer.org/son-propre-resolveur-dns.html>):option forward_upstream '0' dans /etc/config/resolver. (Si, à l'inverse, on veut transmettre à des serveurs aval spécifiques, voir cette discussion sur le forum <https://forum.turris.cz/t/how-to-configure-knot-resolver-to-use-specific-dns-servers/1755/>.)

Mais je voulais surtout une configuration spéciale pour utiliser la racine Yeti <https://www.afnic.fr/fr/ressources/blog/le-projet-yeti-d-experimentation-d-une-racine-dns.html>. Cela nécessite la configuration suivante. D'abord, /etc/config/resolver :

```
config resolver 'common'
        ...
        option keyfile '/etc/kresd/yeti-root.keys'

config resolver 'kresd'
        ...
        option include_config '/etc/kresd/custom.conf'
```

Le fichier des clés de Yeti, indispensable pour la validation DNSSEC, se récupère chez Yeti <https://yeti-dns.org/rootzone.html> (et est réécrit ensuite par Knot, qui gère le RFC 5011). Ensuite, le /etc/kresd/custom.conf contient :

```
hints.root({
        ['bii.dns-lab.net.'] = '240c:f:1:22::6',
        ['yeti-ns.tisf.net .'] = '2001:4f8:3:1006::1:4',
        ['yeti-ns.wide.ad.jp.'] = '2001:200:1d9::35',
        ['dahu1.yeti.eu.org.'] = '2001:4b98:dc2:45:216:3eff:fe4b:8c5b',
        ['dahu2.yeti.eu.org.'] = '2001:67c:217c:6::2',
...
})
```

(Pas grave s'il manque un ou deux serveurs, le "*priming*" - RFC 8109 - s'en occupe.)

Voilà, le résolveur utilise désormais la racine Yeti, comme on peut le vérifier facilement :

```
% dig NS .
...
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 46120
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 26, AUTHORITY: 0, ADDITIONAL: 1
...
;; ANSWER SECTION:
. 86400 IN NS bii.dns-lab.net.
. 86400 IN NS dahu1.yeti.eu.org.
. 86400 IN NS dahu2.yeti.eu.org.
. 86400 IN NS yeti-ns.tisf.net.
. 86400 IN NS yeti-ns.wide.ad.jp.
...
```

Notez que kresd peut être interrogé via une console CLI :

```
# socat - UNIX-CONNECT:/tmp/kresd/tty/$(pidof kresd)
>
```

Et on a droit alors à plein d'informations amusantes (tapez `help()` pour la liste ou bien regardez la documentation <`http://knot-resolver.readthedocs.io/en/latest/`>) :

```
> cache.count()
61495

> cache.stats()
[hit] => 12259583
[delete] => 1
[miss] => 24413638
[insert] => 1542550

> stats.list()
[answer.nxdomain] => 775648
[answer.100ms] => 102752
[answer.1500ms] => 27366
[answer.slow] => 72019
[answer.servfail] => 354445
[answer.250ms] => 125256
[answer.cached] => 3062179
[answer.nodata] => 206878
[query.dnssec] => 80582
[answer.1ms] => 3054309
[predict.epoch] => 27
```

```
[query.edns] => 84111
[predict.queue] => 5946
[answer.total] => 4112245
[answer.10ms] => 77419
[answer.noerror] => 2775274
[answer.50ms] => 393935
[answer.500ms] => 205149
[answer.1000ms] => 47949
[predict.learned] => 447

> cache.get("google.com")
[clients2.google.com] => {
    [CNAME] => true
}
[ghs.google.com] => {
    [CNAME] => true
}
[clients6.google.com] => {
    [CNAME] => true
}
[get.google.com] => {
    [A] => true
}
[accounts-cctld.l.google.com] => {
    [A] => true
    [AAAA] => true
}
[gmail-imap.l.google.com] => {
    [A] => true
}
[inputtools.google.com] => {
    [CNAME] => true
}
[kh.google.com] => {
    [CNAME] => true
}
...
```

Après ces trois « grands dossiers », voici plein de petits détails et de petits projets plus ou moins amusants.

Un des avantages d'un routeur qu'on contrôle complètement, où on est root, c'est qu'on peut tout configurer, y compris les diodes luminescentes, si indispensables à l'informatique. On peut par exemple changer la couleur des diodes selon le débit. La technique en Lua expliquée sur le forum <https://forum.turris.cz/t/led-color-based-on-bandwidth-usage/1262> marche très bien.

Par défaut, le Turris Omnia se gère en HTTPS avec un certificat auto-signé. Même si ce n'est que sur le réseau local, ce n'est pas satisfaisant (et ça empêche les navigateurs de mémoriser les mots de passe, le site n'étant pas considéré comme sûr). Comme je suis utilisateur de CAcert <https://www.bortzmeyer.org/cacert.html>, je voulais utiliser HTTPS avec un certificat CAcert. On le crée (`openssl req -new -nodes -newkey rsa:2048 -keyout server.key -out server.csr -days 1000`), on le fait signer via l'interface Web du CAcert (tout est gratuit et automatique dans CAcert) et on concatène clé privée et certificat (c'est le format qu'attend le serveur HTTPS du Turris, lighthttpd, cf. cet article du forum <https://forum.turris.cz/t/solved-own-ssl-certificate-for-admin-luci-web/1506/>) :

```
# cat server.key  server.pem  > /etc/lighttpd/tls/server.pem
```

Et on change la configuration HTTPS (`/etc/lighttpd/conf.d/ssl-enable.conf`) :

```
$SERVER["socket"] == ":443" {
ssl.engine = "enable"
        ssl.pemfile = "/etc/lighttpd/tls/server.pem"
ssl.use-sslv2 = "disable"
ssl.use-sslv3 = "disable"
}
```

(Et idem avec `$SERVER["socket"] == "[::]:443"` pour IPv6.) En prime, j'active HSTS (RFC 6797) :

```
$HTTP["scheme"] == "https" {
# Add  'HTTP Strict Transport Security' header (HSTS) to sites
setenv.add-response-header  += ( "Strict-Transport-Security" => "max-age=31536000; includeSubDomains" )
}
```

Et je mets une redirection en place depuis HTTP vers HTTPS, dans `/etc/lighttpd/conf.d/https-redirect`

```
$HTTP["scheme"] == "http" {
    # capture vhost name with regex conditiona -> %0 in redirect pattern
        # must be the most inner block to the redirect rule
    $HTTP["host"] =~ ".*" {
            url.redirect = (".*" => "https://%0$0")
   }
}
```

Tout marche bien, désormais.

Un problème fréquent des tunnels (comme celui qu'utilise Free pour faire passer l'IPv6 vers les clients ADSL) est que, la MTU ayant été diminuée, les paquets d'une taille proche de la MTU traditionnelle de 1 500 octets ne passent plus. Cela se voit lorsque ping (avec la taille par défaut <https://www.bortzmeyer.org/ping-taille-compte.html>) ou `openssl s_client` passent mais que HTTP n'arrive pas à faire passer des données. La MTU configurée sur l'Omnia est de 1 480 octets :

```
config interface 'wan'
option ifname 'eth1'
option proto 'dhcp'
option mtu '1480'
```

Désormais, tout passe, mais des machines du réseau local envoient toujours des paquets trop gros (je devrais peut-être diffuser la MTU réduite sur le réseau local). Le routeur note :

```
2016-11-02T07:47:14+00:00 err kernel[]: [  437.388025] mvneta f1034000.ethernet eth1: bad rx status 8fa50000
```

Il y a aussi des problèmes que je n'ai pas réussi à résoudre comme celui d'un accès anonyme aux graphes de trafic <https://forum.turris.cz/t/unauthenticated-access-to-the-web-interface-to-1319>. Notez que je n'ai guère utilisé les forums génériques OpenWrt : les problèmes discutés sont souvent très spécifiques à un modèle de routeur. Par contre, la documentation d'OpenWrt <https://wiki.openwrt.org/doc/start> est très utile si le Turris Omnia est le premier routeur OpenWrt que vous configurez sérieusement. Vous y trouverez notamment des explications sur le système de

configuration <https://wiki.openwrt.org/doc/uci> /etc/config, qui peut être déroutant, si vous venez d'un Unix classique.

Et il y a des problèmes qui sont résolus (le Turris Omnia est en plein développement, et, avec les mises à jour automatiques, on voit des solutions aux problèmes arriver seules). C'est ainsi que le « socat fou <https://forum.turris.cz/t/big-load-socat-stdio-openssl-api-turris-cz-5679/1141> » qui avait fait perdre tant de temps et d'électricité au début de l'Omnia a été réparé <https://gitlab.labs.nic.cz/turris/turris-os-packages/issues/10> sans que j'ai rien eu à faire.

Le routeur Turris permet d'afficher de jolis graphes de trafic (dans LuCI, "*Statistics -¿ Graphs*"). La configuration n'est pas évidente ("*Statistics -¿ Setup*") : j'ai dû créer à la main les répertoites indiqués dans la configuration, puis faire :

```
/etc/init.d/luci_statistics enable
/etc/init.d/collectd enable
```

Par défaut, toutes les données sont perdues à chaque démarrage (voir plus haut la discussion sur la mémoire Flash). On peut changer les répertoires de données pour le disque stable, mais cette modification est perdue à chaque mise à jour du logiciel, hélas. Bref, ce n'est pas encore satisfaisant.