Projet RCPW04

Clément Hertling & Maëldann Le Breton

Table des matières

Introduction

Routeurs Haute Disponibilité

Les abonnements internet professionnels qui permettent d'assurer une haute disponibilité via une redondance d'accès sont souvent assez chers. L'une des solutions consiste à prendre 2 ou plusieurs abonnements. La première partie du compte rendu expliquera notre recherche sur les différentes solutions envisageables, tandis que la deuxième expliquera la mise en oeuvre de notre propre solution.

1 Sources

Partie	Source	Type	Parution	Auteur
1.2, 1.3, 1.4	Réseaux	Livre	2010	Tennenbaum, Wetherall
1.3	DHCP Failover on OpenBSD	Article	2007	Will Backman
1.4	Manuel OpenBSD : pf.conf	Manuel	2016	Projet OpenBSD
2.5	Documentation VLANs Cisco	Documentation	N/A	Cisco
2.5	Documentation Trunks Cisco	Documentation	N/A	Cisco
2.5	nixCraft : FreeBSD VLANs	Documentation	2009	VIVEK GITE
2.6.1	VLAN setup on OpenBSD	Article	2011	N/A
2.6.2	PF User Guide	Documentation	N/A	Projet OpenBSD
2.6.2	Manuel OpenBSD : pf.conf	Manuel	2016	Projet OpenBSD
2.6.2	Handbook FreeBSD : PF	Documentation	N/A	John Ferrell
2.6.3	Manuel FreeBSD : pfsync	Manuel	2011	N/A
2.6.3, 2.6.4	Documentation OpenBSD : pfsync	Documentation	N/A	N/A
2.6.4	Handbook FreeBSD : CARP	Documentation	N/A	Allan Jude
2.6.5	undeadly.org : User Stories	Article	2007	“sean”

Première partie

Compte Rendu de Projet

Chapitre 1
Recherche Documentaire

1.1 Introduction

Ce projet est constitué de plusieurs parties : Il nous faut tout d'abord concevoir une architecture répondant aux demandes formulées :

- Répartition de charge (Multihoming),
- Basculement Automatique (Failover),
- et Hiérarchisation des flux (QoS),
- le tout avec du matériel peu couteux

Nous allons ici expliquer en quoi chacune de ces parties consiste en expliquant lorsque c'est possible les détails des protocoles et/ou des systèmes à mettre en place, et nous donnerons quelques exemples de technologies répondant au problème en question.

1.2 Répartition de charge

La répartition de charge, ou multihoming, est une technologie qui permet de connecter un réseau final à plusieurs réseaux de transport et de transmettre des flux réseaux en utilisant ces différents réseaux de transport. La répartition de charge peut être mise en place à plusieurs niveaux du réseau : au niveau 3, on l'appelle multihoming ou routage multipath, au niveau 4, multipath `TCP` ou multipath `UDP` et aux niveaux supérieurs le nom du système dépend de l'implémentation, aucun standard, soit-il informel, n'existe.

Le système qui nous intéresse pour cette maquette est bien évidemment le multihoming, de niveau 3 donc, puisque nous gérons ici des routeurs. Le principe est simple : le système d'exploitation sur lequel le routeur est basé gère une table de routage, en général au niveau du kernel. Ici, cette table contient deux entrées pour une même destination, ce qui permet de router un même paquet par deux routes différentes. Le choix de la route effectivement choisie est effectuée par un algorithme de répartition de charge, le plus utilisé étant probablement round-robin, qui consiste à répartir équitablement les paquets.

On peut préciser qu'en réalité, les décisions de routage ne sont pas prise pour chaque paquet mais bien pour chaque flux, pour éviter les problèmes avec les connexions `TCP` notamment. La RFC 2992 est une méthode standardisée qui permet à ce système de fonctionner, en utilisant des hash `CRC` dans le header du paquet pour identifier le flux.

L'utilisation du multihoming offre plusieurs avantages, dont une amélioration en générale conséquente du débit offert à chaque hôte sur le réseau (le débit maximal d'une connexion ne change pas, mais le débit total disponible aux hôtes est doublé (ou triplé, etc.) et chaque connexion est donc en moyenne plus rapide si le réseau était single homed), mais aussi une amélioration de la latence des connexions, et une possibilité de failover. Elle présente cependant quelques inconvénients : le routeur nécessite un CPU plus efficace pour router les paquets, il est nécessaire de configurer une méthode de failover dans le cas ou l'un des uplinks tombe sous peine de voir la moitié des connexions perdues.

Puisque cette technologie est en réalité une façon spéficique de faire du routage, elle fait partie du kernel de certains OS, et il n'existe donc pas d'implémentation "détachée" d'un OS en particulier. On peut cependant citer certains des OS qui implémentent le routage multipath, notamment Linux ou OpenBSD, en plus des systèmes fonctionnant sur certains routeurs de grands constructeurs d'équipement réseau.

1.3 Basculement Automatique

Le basculement automatique, ou failover, est une expression désignant la situation où un service est proposé par plusieurs serveurs à la fois, en coordination, de façon à ce qu'en cas de problème sur l'un d'entre eux, l'autre puisse prendre le relais sans interruption de service. Cette situation est applicable à de nombreux domaines et à de nombreux types de services, et ne sert pas que dans l'informatique : lorsque la SNCF remplace des trains par des bus, c'est en quelque sorte du failover, même si ça n'est pas automatique. Dans notre cas, nous utilisons du failover pour la plupart des services que nous mettons en place, puisque l'une des orientations du projet est la résilience du réseau. Nous utilisons pour cela plusieurs technologies, à la fois au niveau du routage et des services de plus haut niveau (firewall, DHCP,…).

Il est compliqué de citer une technologie de failover particulière, puisque le concept s'applique à de nombreux domaines, mais nous pouvons citer certaines méthodes concernant les services pour lesquels nous avons mis en place du failover.

Tout d'abord, pour le routage, il existe plusieurs protocoles permettant à plusieurs hôtes de partager la même adresse IP sur un même réseau, ce qui permet à un hôte de prendre le relai lorsqu'un autre cesse de fonctionner, mais aussi de répartir les paquets entre les différents hôtes partageant cette adresse. On peut citer par exemple VRRP, un protocole standard créé par l'IETF, ou CARP, un protocole centré sur la sécurité créé par le projet OpenBSD. Cisco a lui aussi son protocole nommé HSRP.

Pour la gestion des routes proprement dites, il est nécessaire d'avoir un mécanisme permettant à un hôte de supprimer une route si elle ne mène nulle part. Dans notre cas, un routeur dont l'uplink ne fonctionne plus pour une raison ou une autre doit être capable de le détecter et de se mettre à router les paquets qu'il

reçoit vers l'autre routeur. Il n'existe pas de véritable protocole pour cette tâche, qui est triviale. Il existe cependant quelques logiciels permettant de faire cela, dont ifplugd ou netplug sous linux, et ifstated sous BSD.

En ce qui concerne le firewall, il n'existe pas de protocole de synchronisation des règles (ce qui est logique, même sur un même réseau, deux firewalls ont très probablement besoin de règles différentes), mais il existe des protocoles, standardisés ou non, qui permettent de synchroniser les tables d'état de ces firewalls. Sous Linux/Netfilter, on peut utiliser `Conntrack`, tandis que sous OpenBSD et NetBSD, avec `PF`, on peut utiliser pfsync. Cisco, encore une fois, a son propre protocole, nommé `SSO` (**S**tateful **S**witch**O**ver).

Enfin, nous nous devons de fournir un service de `DHCP` sur le réseau interne. Il existe plusieurs méthodes permettant d'avoir deux serveurs `DHCP` sur un même réseau : par exemple, il est possible de partager (via `NFS`, `iSCSI`, ou tout autre protocole de partage de fichiers) la base de donnée du serveur `DHCP`, ce qui permet aux deux serveurs de savoir quelles adresses ont déjà été offertes. En introduisant un léger délai dans l'un des serveurs, on peut alors s'assurer que seul un des deux serveurs répondra lorsqu'une requête arrive : le serveur primaire lorsqu'il est disponible, le serveur secondaire sinon. Le problème avec cette méthode est que le partage de fichiers doit être hébergé quelque part, mais qu'il ne peut pas l'être sur l'un des deux routeurs : si c'était le cas, lorsque ce serveur rencontre un problème, la redondance ne fonctionnerait pas, puisque l'autre n'a plus accès à la base de données. Or, nous n'avons pas de troisième machine à utiliser simplement pour un partage de fichiers. Une autre méthode disponible est de diviser l'espace d'adressage entre deux serveurs `DHCP` et encore une fois d'introduire un délai sur l'un des serveurs. L'inconvénient ici est que nous perdons 50% de l'espace d'adressage de notre réseau, ce qui est un problème dans notre cas.
Nous nous sommes alors rendu compte qu'il existe dans le serveur `DHCP`, fourni par l'`ISC`, une option pour synchroniser la base de données d'adresses offertes, entre autres, et qui permet à deux serveurs de ne pas faire de réponses en double.

1.4 Hiérarchisation des Flux

La Hiérarchisation des Flux, aussi appellée Qualité de Service (QoS, Quality of Service en anglais), est un terme désignant un ensemble de techniques permettant de séparer les flux de données traversant un réseau, et de créer des règles permettant d'en prioriser certains au détriment d'autres. Cette méthode permet notamment de définir des canaux fixes pour certains usages, comme la téléphonie IP, qui permettent de s'assurer que les communications utilisant cette technologie ne rencontreront pas de problèmes à cause d'une autre utilisation du réseau. On peut aussi l'utiliser pour définir des règles dynamiques, qui s'ajustent en fonction des différents types de données qui passent sur le réseau.

La QoS peut être mise en place via différentes méthodes, mais la plus courante

est de faire en sorte que les routeurs, qu'ils soient internes au réseau ou edge, appliquent de la QoS en fonction des règles paramétrées dans leurs firewalls. Cela peut être fait avec tous les firewalls modernes, sous Linux avec Netfilter, sous BSD avec pf, ou sur les équipements réseaux dédiés avec leur firewall intégré.

Chapitre 2

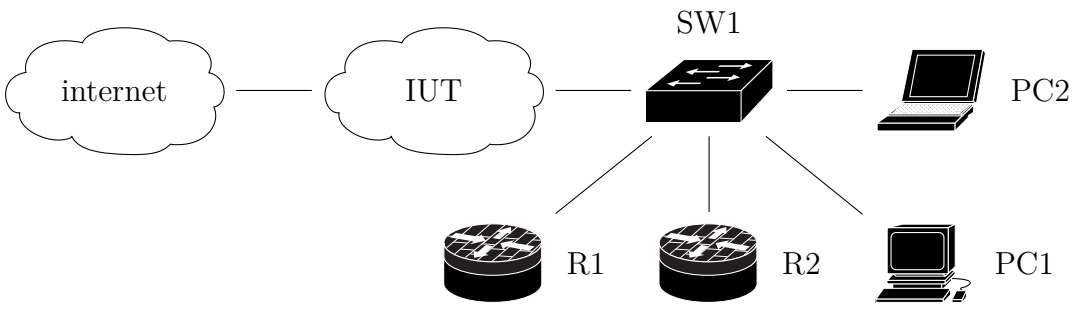
Réalisation

2.1 Introduction

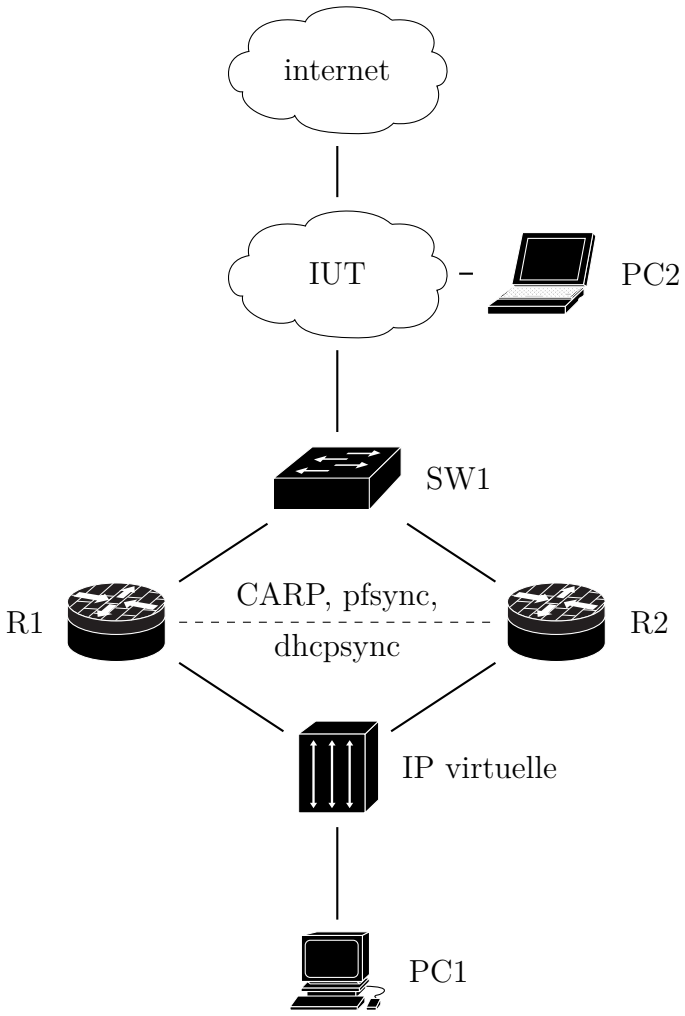
Le projet consiste à mettre en place un réseau qui rencontre plusieurs critères : il doit à la fois être redondant, mettre en action de la répartition de charge, et une hiérarchisation des flux pour assurer le passage du traffic VoIP. Le tout doit être réalisé sur un budget limité.

Nous utiliserons deux "routeurs" basés sur des machines grand public, fonctionnant sur un système gratuit et open-source (OpenBSD et FreeBSD, comme expliqué plus loin), chacun étant connecté à un réseau externe fourni par un opérateur. Nous simulerons le réseau interne avec un autre ordinateur. Les routeurs étant basés sur du matériel grand public, ils fonctionneront en tant que Router-on-a-Stick et nous utiliserons donc un switch pour séparer les réseaux avec des `VLANs`. Le réseau de l'IUT sera utilisé pour connecter notre maquette à Internet, chaque opérateur étant simulé par un câble Ethernet différent. Nous utiliserons de nombreux logiciels et protocoles open-source pour gérer la répartition de charge et la redondance de cette maquette, dont entre autres `CARP`, le serveur `isc dhcpd`, `pfsync`, etc.
Nous avons encontré au fil de notre réalisation quelques écueils, que nous détaillerons ci-dessous. La réalisation de ce projet fut une expérience enrichissante, et nous avons progressé dans notre compréhension des systèmes à haute disponibilité et des systèmes `UNIX` non-Linux.

2.2 Diagramme Physique



2.3 Diagramme Réseau



2.4 Présentation générale

- **Materiel**
  Pour ce projet, nous utilisons un matériel limité :
  - **R1**, un routeur sous OpenBSD fonctionnant sur l'un des ordinateurs portables de l'IUT. Il est le routeur principal.
  - **R2**, un routeur sous FreeBSD fonctionnant sur une Cubieboard 2, une plateforme de développement ARM. Ce routeur est sous FreeBSD car il est impossible d'installer OpenBSD sur cette plateforme de développement. Il est le routeur secondaire.
  - **SW1**, un switch cisco fourni par l'IUT. Ce dernier fonctionne sous Cisco IOS version 12.1.
  - **Quelques cables Ethernet**

- **Configuration du switch**
  Nous avons commencé par configurer le switch de façon à séparer les différentes machines en plusieurs réseaux via des `VLANs`.

  - **Trunk**
    Nous avons créé deux trunks pour séparer les routeurs tout en les laissant intéragir avec différents réseaux.
    `R1` se trouve sur le port `fa0/1` et `R2` sur `fa0/2`.

  - **VLAN**
    Nous utilisons 3 `VLANs`, chacun représentant un réseau différent.
    Le `VLAN` du réseau interne fait partie des deux trunks, alors que les `VLANs` des deux réseaux opérateurs sont seulement présents dans le trunk de leur routeur respectif.
    Le `VLAN` du réseau interne occupe les ports `fa0/3` à `fa0/22`. Celui du premier opérateur occupe le port `fa0/23` alors que le deuxième opérateur se trouve sur le port `fa0/24`.

- **Configuration des routeurs**
  Les routeurs étant le cœur du sujet, leurs configurations demandent plus d'étapes.

  - **Interfaces VLAN**
    Puisque les routeurs sont connectés à des ports trunk, ils doivent tagger leurs paquets pour être capables d'envoyer des paquets sur les deux réseaux de destination.

  - **Packet Filter**
    Nous avons utilisé `PF` pour permettre aux machines du réseau interne d'accéder à Internet et de se connecter en ssh aux routeurs tout en bloquant le trafic indésirable.

  - **Pfsync**
    Pour synchroniser l'état des pare-feu nous avons utilisé `Pfsync`.

  - **Ifstated**
    Pour enlever ou rajouter la route de `R1` vers `R2` en fonction de l'état de `R2`, nous avons utilisé ifstated.

  - **CARP**
    `CARP` nous permet de créer une adresse virtuelle, partagée par les deux routeurs, qui envoie les requêtes reçues au routeur maître ou au secondaire s'il n'est pas disponible.

  - **DHCPD**
    Nous utilisons le serveur dhcp de l'`ISC` pour partager le rôle de serveur `DHCP` entre leurs deux routeurs.

2.5 Configuration du switch

Pour simuler le réseau d'un hôpital, nous devons configurer notre switch pour créer différents réseaux :

- **le réseau interne** : c'est le réseau prévu pour les utilisateurs de l'hôpital. Il est relié par les deux routeurs à Internet.
- **les réseaux opérateurs** : ils représentent les deux opérateurs de l'hôpital, qui sont reliés au réseau de l'IUT pour accéder directement à Internet.

2.5.1 Trunk

Nous utilisons des trunks pour permettre aux routeurs d'être présents sur deux réseaux différents, le réseau interne et leurs réseaux opérateurs respectifs (`R1` sur le premier réseau opérateur, `R2` sur le deuxième), de façon à transmettre les paquets de l'un à l'autre.

Les trunks permettent donc de relier les deux réseaux opérateurs au réseau interne tout en laissant les routeurs filtrer les paquets entre eux.
Nous utilisons les commandes suivantes pour configurer les trunks :

- **Pour R1**, qui se trouve sur le port `fa0/1` :
```
interface fa0/1
    no shutdown
    switchport mode trunk
    switchport trunk native vlan 100
    switchport trunk allowed vlan none
    switchport trunk allowed vlan add 300,400
exit
```
- **Pour R2**, qui se trouve sur le port `fa0/2` :
```
interface fa0/2
    no shutdown
    switchport mode trunk
    switchport trunk native vlan 200
    switchport trunk allowed vlan none
    switchport trunk allowed vlan add 300,500
exit
```

2.5.2 VLANs

Les différents `VLANs` nous permettent de mettre en place les différents réseaux dont nous avons besoin pour simuler le réseau d'un hôpital.
Nous avons trois `VLANs` pour les trois différents réseaux :

- **300** : pour le réseau interne, allant du port `fa0/3` au port `fa0/22`. La plage d'adresse de ce `VLAN` est `30.30.30.0/24`.
  Il est configuré par ces commandes :
```
vlan 300
    name vInt
exit
in fa0/3 - 22
    no shutdown
    switchport mode access
    switchport access vlan 300
exit
```
- **400** : pour le premier réseau opérateur, occupant le port `fa0/23`. Les adresses des machines de ce `VLAN` sont configurées par le serveur `DHCP` de l'IUT.
  Il est configuré par ces commandes :
```
vlan 400
    name vOpA
exit
in fa0/23
    no shutdown
    switchport mode access
    switchport access vlan 400
exit
```
- **500** : pour le deuxième réseau opérateur, occupant le port `fa0/24`. Les adresses des machines de ce `VLAN` sont aussi configurées par le serveur `DHCP` de l'IUT.
  Il est configuré par ces commandes :
```
vlan 500
    name vOpB
exit
in fa0/24
    no shutdown
    switchport mode access
    switchport access vlan 500
exit
```

2.6 Configuration des routeurs

Les routeurs étant le cœur de la réalisation, ils demandent plus de configurations pour respecter les besoins de la maquette.
Nous devons arriver en un premier temps à créer des interfaces de VLAN, qui permettent aux deux routeurs de tagger leurs paquets avec un en-tête 802.1Q et donc d'émettre sur les deux VLANs auxquels ils ont accès (et de recevoir les paquets taggés, en les désencapsulant).

Les routeurs doivent ensuite transférer les paquets entre le réseau interne et les réseaux opérateurs grâce à un NAT. Ils doivent aussi laisser passer les paquets ssh permettant de les configurer à distance.
Les routeurs doivent enfin communiquer ensemble pour décider en temps réel de la répartition des paquets de façon à utiliser les deux connexions Internet en même temps (on fait alors du routage multipath).

2.6.1 Interfaces de VLANs

Puisque nos réseaux sont séparés en VLANs, et que les routeurs sont sur des ports trunks, ces derniers recevront les paquets de tous les VLANs autorisés sur leurs trunks respectifs, mais ils les recevront encore encapsulés dans le protocole 802.1Q. Pour qu'ils soient capables de décoder l'information qu'ils reçoivent, nous avons donc besoin de paramétrer des interfaces de VLANs sur ces routeurs.

Le routeur R1 a comme adresse 30.30.30.1/24 pour le réseau interne et reçoit son adresse dynamiquement par le serveur DHCP de l'IUT pour son réseau opérateur.
Quant au routeur R2, son adresse est 30.30.30.2/24 pour le réseau interne et il reçoit aussi son adresse dynamiquement pour son réseau opérateur. Les configurations des interfaces physiques et virtuelles pour R1 sont :

- **bge0** :
  /etc/hostname.bge0
  ```
  up
  ```
  Il est nécessaire de préciser que l'interface physique est activée, sinon les interfaces de vlan ne fonctionneraient pas.
- **vlan0** :
  /etc/hostname.vlan0
  ```
  inet 30.30.30.1 255.255.255.0 30.30.30.255 vlan 300 vlandev em0
  ```
  Cette interface est celle qui est présente dans le réseau interne.
- **vlan1** :
  /etc/hostname.vlan1
  ```
  dhcp vlan 400 vlandev em0
  ```
  Cette interface est celle présente sur le réseau externe, elle est configurée par DHCP.
- **carp0** :
  /etc/hostname.carp0
  ```
  vhid 125 pass vitrygtr carpdev vlan0 advbase 3 advskew 1
    state master 30.30.30.254 netmask 255.255.255.0
  ```
  L'interface carp0 nous sert à mettre en place du failover et de la répartition de charge entre les deux routeurs. Le routeur R1 est ici déclaré maître, puisqu'il peut faire du routage multipath, contrairement à R2.

La configuration pour R2 est :

/etc/rc.conf

```
ifconfig_dwc0="up"
```

```
vlans_dwc0="300 500"
ifconfig_dwc0_300="inet 30.30.30.2 netmask 255.255.255.0"
ifconfig_dwc0_300_alias0="vhid 125 advbase 3 advskew 200 \
  state backup pass vitrygtr alias 30.30.30.254/24"
ifconfig_dwc0_500="DHCP"
```

Cette configuration fait partie du fichier `/etc/rc.conf`, que nous avons ici découpé en plusieurs parties pour simplifier la compréhension des différents paramètres inclus.

2.6.2 Packet Filter

En tant que firewall, nous utilisons Packet Filter (`PF`), le pare-feu de `FreeBSD` et `OpenBSD`. Il nous permet non seulement de transférer les paquets que nous voulons garder, en redirigeant et en nattant les connexions du réseau interne vers Internet via l'un des réseaux opérateurs mais aussi de laisser passer les paquets ssh pour la configuration à distance. Nous laissons aussi passer les paquets de `pfsync` et `CARP`, permettant la communication entre les deux routeurs.

La qualité de service garantissant la prioritisation de la `VoIP`, permettant de garder des communications stables pendant une surchage du réseau, est aussi gérée par `PF`.
Enfin, les paquets `DHCP` sont aussi transférés correctement pour la configuration des adresses IP des machines du réseau interne. La configuration pour `R1` est la suivante :
`/etc/pf.conf`

```
set skip on lo

# définition des variables
int="30.30.30.0/24"
int_alt="50.50.50.0/24"
ext="0.0.0.0/0"
ip_int="30.30.30.1"
int_if="vlan0"
ext_if="vlan1"

# defaut : bloquage
block all

# vérification des paquets, anti-spoofing
antispoof for $int_if
antispoof for $ext_if

# nous laissons passer les pings pour cette maquette
pass proto icmp
```

```
# nous mettons en place le NAT de l'interieur vers Internet
pass in on $int_if from $int to any keep state
pass out on $ext_if from $int to $ext nat-to $int_if keep state

# carp, pfsync et dhcpsync
pass out on $int_if proto carp keep state
pass quick on $int_if proto pfsync keep state
pass in on $int_if proto udp to any port 8067 keep state
pass out on $int_if proto udp to any port 8067 keep state

# QoS (le port 4000 sert pour la démonstration)
queue int on $int_if bandwidth 100M
queue sip parent int bandwidth 10M min 5M max 15M
queue other parent int bandwidth 90M min 85M default
pass in on $int_if proto { tcp udp } from any to any \
    port { 5060 5061 5064 5065 } set queue sip
pass out on $int_if proto { tcp udp } from any to any \
    port { 5060 5061 5064 5065 } set queue sip
pass in on $int_if proto tcp from any to any port 4000 set queue other
pass out on $int_if proto tcp from any to any port 4000 set queue other

# nous laissons passer les connexions SSH vers le routeur
pass in on $int_if proto tcp from $int to $ip_int port ssh keep state
pass out on $int_if proto tcp from $ip_int port ssh to $int keep state
```

La configuration pour `R2` diffère quelque peu de `R1` puisque les versions de `PF` ne sont pas identiques sur `FreeBSD` et `OpenBSD`. De plus, le PF de FreeBSD ne supporte pas la QoS avec l'interface réseau qui est présente sur notre routeur, nous ne pouvons donc pas faire de QoS avec `R2`.

`/etc/pf.conf`

```
set skip on lo

# définition des variables
int="30.30.30.0/24"
ext="0.0.0.0/0"
ip_int="30.30.30.2"
int_if="dwc0.300"
ext_if="dwc0.500"

défaut : bloquage
block all

# vérification des paquets, anti-spoofing
antispoof for $int_if
```

```
antispoof for $ext_if

# nous laissons passer les pings pour cette maquette
pass proto icmp

# nous mettons en place le NAT de l'interieur vers Internet
nat on $ext_if from $int to any -> ($ext_if)
pass in on $int_if from $int to any keep state
pass out on $ext_if from any to $ext

# carp, pfsync et dhcpsync
pass out on $int_if inet proto carp keep state
pass quick on $int_if inet proto pfsync keep state
pass in on $int_if inet proto udp to port 8067 keep state
pass out on $int_if inet proto udp to port 8067 keep state

# pas de QoS

# nous laissons passer les connexions SSH vers le routeur
pass in on $int_if inet proto tcp from $int to $ip_int \
    port ssh keep state
pass out on $int_if inet proto tcp from $int to $ip_int \
    port ssh keep state
```

2.6.3 Pfsync

Pour garder les connexions entre le réseau interne et Internet, en cas de problème avec l'un des routeurs, nous devons synchroniser l'état des pare-feu entre les deux routeurs. Pour ce faire, nous utilisons le protocole pfsync, qui permet de synchroniser les tables d'états de deux pare-feu `PF`.
Pour `R1`, la configuration est :
`/etc/hostname.pfsync0`

```
syncdev vlan0 syncpeer 30.30.30.2
```

Quant à `R2`, sa configuration est :
`/etc/rc.conf`

```
pfsync_enable="YES"
pfsync_syncdev="dwc0.300"
pfsync_syncpeer="30.30.30.1"
```

2.6.4 CARP

Le protocole CARP (Common Address Redundancy Protocol) sert à faire en sorte que nos deux routeurs partagent la même adresse, ce qui permet d'avoir un

failover quasi immédiat si l'un des deux cesse de fonctionner. Ici, nous faisons en sorte que `R1` et `R2` partagent l'adresse `30.30.30.254`, avec `R1` en primaire, et `R2` en secondaire, puisque `R1` supporte le routing multipath, et qu'il est capable d'envoyer des paquets vers `R2`, tandis que ce dernier ne peut pas faire de même. Désigner `R1` comme routeur primaire permet ainsi de répartir la charge entre les deux routeurs.

La configuration de CARP pour R1 est telle que suit :
/etc/hostname.carp0

```
vhid 125 pass vitrygtr carpdev vlan0 advbase 3 advskew 1
  state master 30.30.30.254 netmask 255.255.255.0
```

Et pour R2 :
/etc/rc.conf

```
ifconfig_dwc0_300_alias0="vhid 125 advbase 3 advskew 200 \
  state backup pass vitrygtr alias 30.30.30.254/24"
```

2.6.5 Ifstated

Le load-balancing des paquets peut être fait grâce à une route multihoming, depuis R1 à R2, puisque CARP envoie tous les paquets au maître, qui devrait être R1 si il n'y a pas de problème. Si R1 ne fonctionne plus, R2 recevra tous les paquets directement depuis CARP.
Cependant, nous devons enlever ou rajouter une route sur R1 en fonction de l'état de R2. Pour ce faire, ifstated ping à intervalle régulier R2 pour savoir s'il est encore opérationnel. S'il ne l'est pas, sa route est supprimée, sinon elle est gardée ou ajoutée si besoin.
Sa configuration est :
/etc/ifstated.conf

```
state auto {
   if $peer
         set-state multihome
   if ! $peer
      set-state singlehome
}

state multihome {
   init {
      run "route add -mpath default 30.30.30.2"
   }

   if ! $peer
      set-state singlehome
}

state singlehome {
   init {
      run "route delete default 30.30.30.2"
   }
   if $peer
```

```
        set-state multihome
}

init-state auto
```

2.7 Problèmes

Le matériel que nous avons utilisé pour le projet nous semblait être une manière intéressante de tester le support multiplateforme de notre solution, et de trouver une manière pour l'hôpital de réduire encore les coûts : une board de développement telle que celle que nous avons utilisée (Cubieboard 2) ne coûte qu'une quarantaine d'euros. Cependant, cette board n'est pas actuellement supportée par OpenBSD, et n'est que mal supportée par FreeBSD : vers la fin de notre réalisation, nous avons eu de nombreux problèmes avec `pfsync`, qui causait des kernel panics sur la board. Cela était dû à la version de FreeBSD installée, qui se trouvait être une version de développement, mais qui était aussi la seule version pour laquelle une distribution binaire était disponible : toutes les autres versions de FreeBSD demandaient une cross-compilation depuis les sources.

Ne pouvant pas terminer le projet avec ce problème, nous avons donc décidé de migrer notre routeur `R2` sur un deuxième ordinateur portable, sous OpenBSD lui aussi. La migration fut simple, puisque nous avions déjà un routeur sous ce système. Notre maquette fonctionne désormais donc avec deux routeurs sous OpenBSD. `R2` est donc finalement lui aussi capable de faire de la QoS.

Annexes

Glossaire

TCP Transport Control Protocol, protocole réseau de couche transport (4) offrant des garanties de fiabilité, et opérant en mode connecté.

UDP User Datagram Protocol, protocole réseau de couche transport (4) n'offrant aucune garantie de fiabilité mais plus léger que TCP.

Multihoming Technologie réseau de niveau trois, permettant à un routeur de posséder deux routes vers un même réseau cible.

RFC Request for Comments, publications de l'IETF contenant en général des standards utilisés par la plupart des acteurs d'Internet.

IETF Internet Engineering Task Force, groupe d'acteurs d'Internet ayant pour but de définir les technologies utilisées sur les réseaux.

CRC Cyclic Redundancy Check, code de détection d'erreur courant permettant de détecter les corruptions de fichiers et/ou de transferts.

failover Basculement, ensemble de technologies permettant à un service rencontrant un problème de continuer à fonctionner.

CPU Central Processing Unit, nom communément donné à un processeur, ou à un cœur d'un processeur.

uplink Dans un réseau, lien vers un réseau plus grand. En général utilisé pour parler du lien vers Internet.

kernel Partie d'un OS qui fait le lien entre le Hardware et le Software.

OS Operating System, Système d'Exploitation. Ensemble logiciels fonctionnant sur une machine, sur lesquels sont basées les applications utilisateur.

Hardware Le Hardware, par opposition au Software (logiciel) et au Middleware (micrologiciel), représente le matériel physique (processeur, mémoire, disque dur, etc.) composant un système informatique.

Firewall Pare-Feu, outil de traitement des paquets ayant pour but principal de filtrer certains types de trafics.

SNCF Société Nationale des Chemins de Fer, entreprise française proposant des services de transport en train.

DHCP Dynamic Host Configuration Protocol, protocole réseau visant à configurer les hôtes automatiquement lors de leurs connexions.

firewall pare-feu, outil de sécurité permettant de bloquer certaines connexions et de traiter les paquets reçus par une machine de différentes manières.

NFS Network File System, service réseau donnant accès à un système de fichier via le réseau.

iSCSI Internet Small Computer Systems Interface, protocole permettant d'exécuter des commandes SCSI directement via une connexion IP.

ISC Internet Systems Consortium, groupe de personne travaillant dans les réseaux œuvrant pour créer des logiciels.

QoS Quality of Service, pratique de prioriser certains flux pour permettre une meilleure qualité de services à certains usages.

VoIP Voice over IP, protocoles permettant d'effectuer des appels téléphoniques via Internet.

UNIX OS datant du millieu des années 80, présent maintenant dans de nombreux domaines et sur de multiples architectures via ses clones Open-Source.

Logs Journaux d'évènements. Fichiers contenant les informations émises par les logiciels pendant leur fonctionnement. Ils permettent d'établir des diagnostics lors des erreurs, ou bien de détecter des comportements anormaux, par exemple.