Lothaire

CRIL Centre Interuniversitaire de Ressources Informatiques de Lorraine

Keepalived

Haute disponibilité et répartition de charge enfin libérées !

JRES'2011 Toulouse

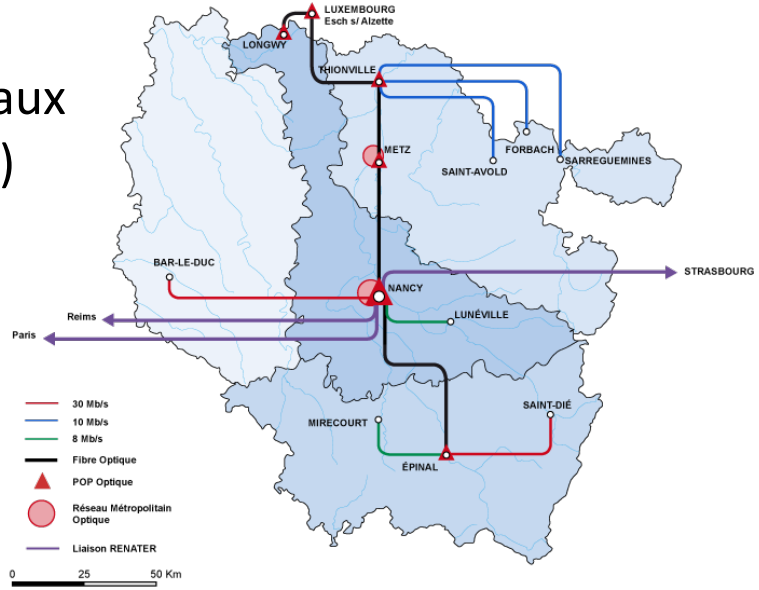
Alexandre SIMON
Alexandre.Simon [AT] ciril.fr

Au programme

- Contexte
- *Linux* et *Netfilter*
- *IPVS*
- *Keepalived*
- La quadrature du répartiteur
- Exemples
- Conclusion

Contexte

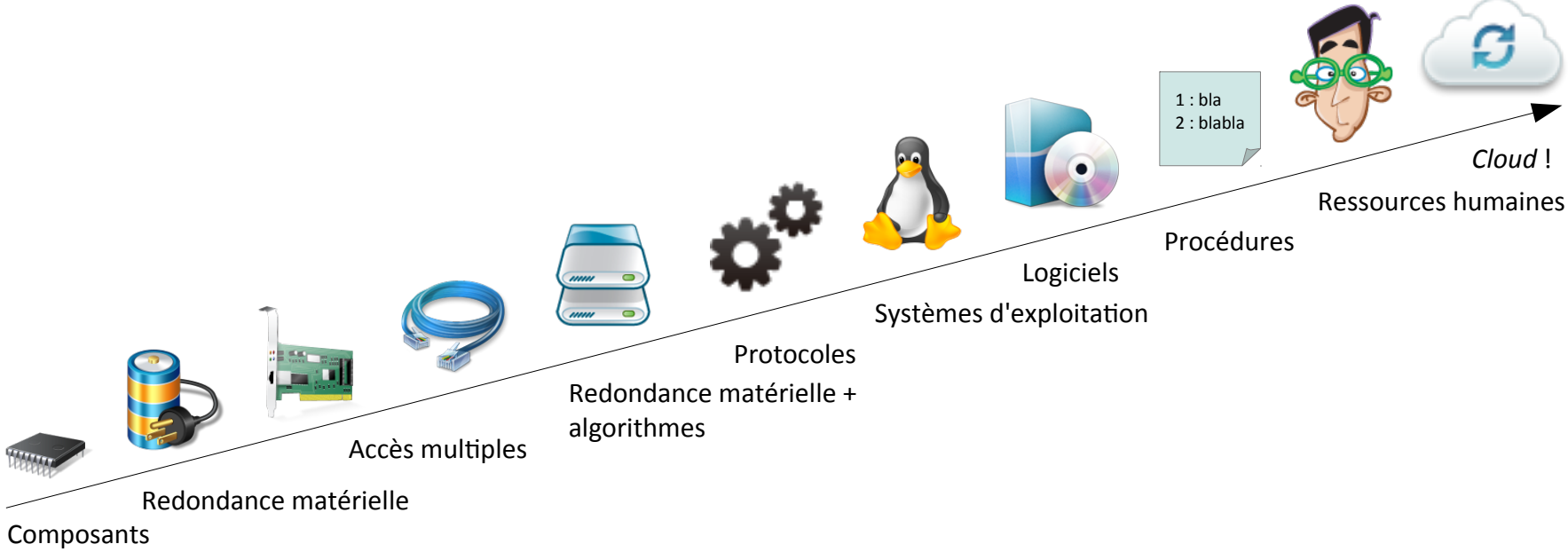
- Équipe réseau du CIRIL
  - exploite, administre et supervise les réseaux lorrains (Lothaire, StanNet et AmpèreNet)
    - 13 villes connectées
    - 93 sites raccordés
    - plus de 1300 équipements réseaux
  - déploie des services réseaux « avancés »
    - portail captif YaCaP
    - filtrage Web iWash (*cf*. poster JRES'2011)



- Développement d'architectures « maison » hautement disponibles
  - assemblage d'outils libres
  - recherche du meilleur compromis « ressources utilisées / service rendu »

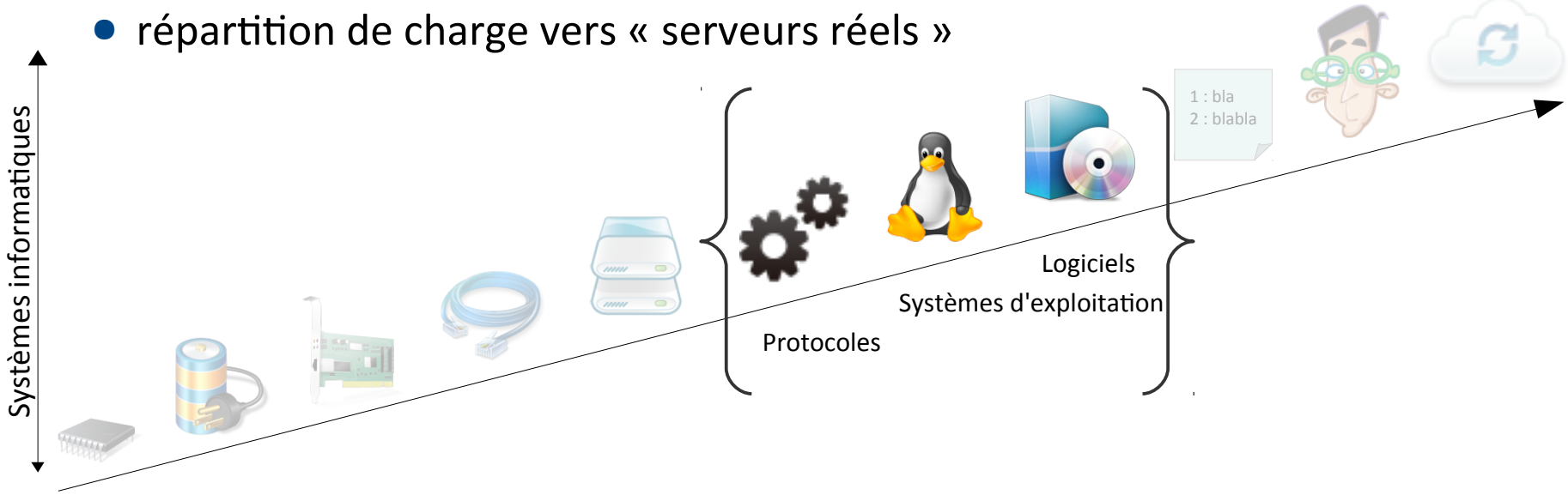
Contexte

- Haute disponibilité
  - recherche *Google* « *high availability* » → 100 millions de réponses !
  - la haute disponibilité : elle est partout, à tous les niveaux !



Contexte

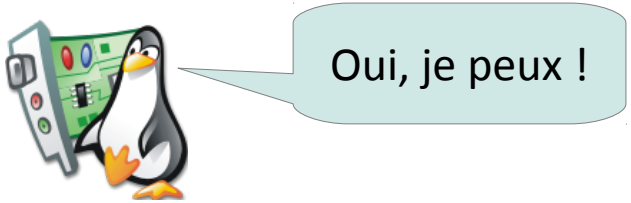
- Dans cette présentation :
  - « Haute disponibilité » = haute disponibilité réseau niveau 3 et 4
  - virtualisation de services réseaux *TCP/UDP IPv4/IPv6*
  - répartition de charge vers « serveurs réels »



Au programme

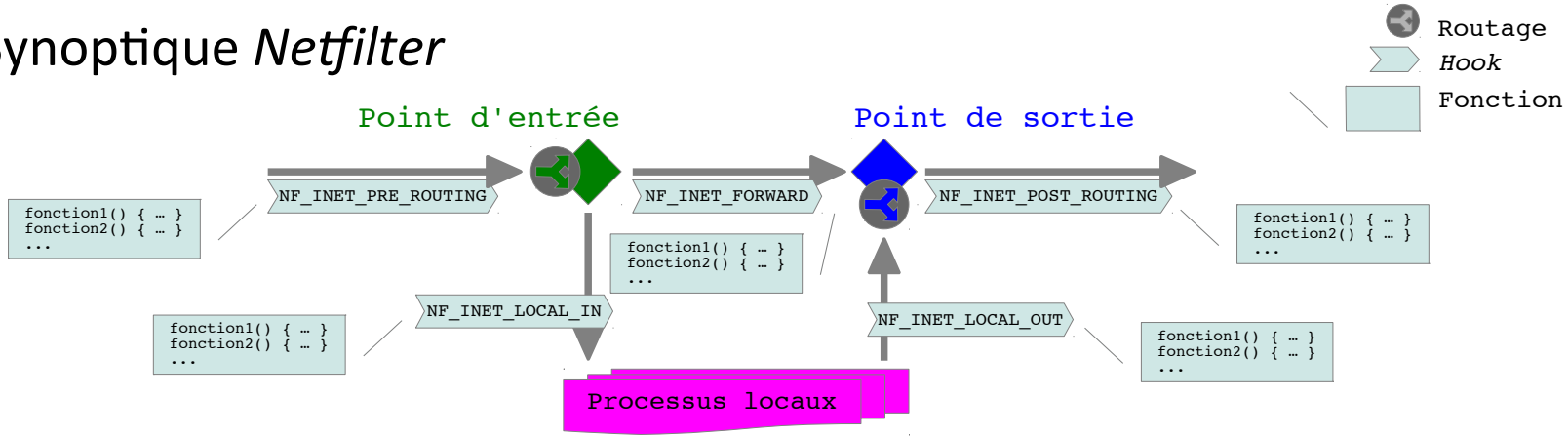
- Contexte
- *Linux* et *Netfilter*
- *IPVS*
- *Keepalived*
- La quadrature du répartiteur
- Exemples
- Conclusion

Linux pour la mise en œuvre d'applications hautement disponibles ?



- *Framework Netfilter*
  - définit la pile réseau
  - décrit et implémente la gestion des flux des paquets réseau
  - propose des points d'accroche (*hooks*) pour intercepter et manipuler les paquets réseau

- Synoptique *Netfilter*



- Qui utilise *Netfilter* ?

  - *ip[6]_tables*, *arp_tables*, *ebtables*, *conntrack*, …
  - *ipvs* : virtualisation service réseau et répartition de charge vers serveurs

Au programme

- Contexte
- *Linux* et *Netfilter*
- *IPVS*
- *Keepalived*
- La quadrature du répartiteur
- Exemples
- Conclusion

- Recherche *Google* « *IPVS* » → *http://www.linuxvirtualserver.org*
- *LVS - Linux Virtual Server*
  - projet libre piloté par *Wensong ZHANG*
  - objectif : « Développement d'une solution avancée, robuste et performante de virtualisation de service réseau et de répartition de charge sous *Linux* »
    - *IPVS – IP Virtual Server* : répartition de charge *IP* (niveau 3 et 4)
    - *KTCPVS - Kernel TCP Virtual Server* : répartition de charge applicatif (niveau 7)
    - composants pour la gestion d'ensemble de serveurs (cluster)

LVS décrit les principes et les outils pour le déploiement de solutions de haute disponibilité réseau.
IPVS implémente une solution de virtualisation et de répartition de charge *IP*.

- Fiabiliser un service = redonder le service sur plusieurs serveurs
  - les arrêts de serveurs sont normaux (ou pas!)
  - un service fiabilisé doit être « présent » ou « migrable » sur plusieurs serveurs

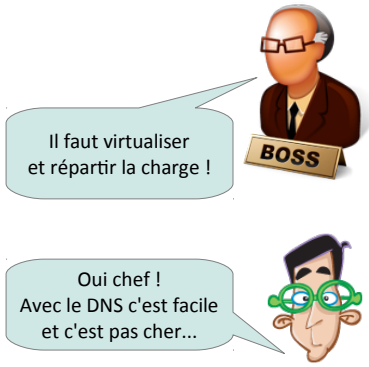
- Virtualisation et répartition de charge à base de *DNS*

```
reseau.ciril.fr.    IN   A    tic.ciril.fr.
                    IN   A    tac.ciril.fr.
```

réponses alternatives

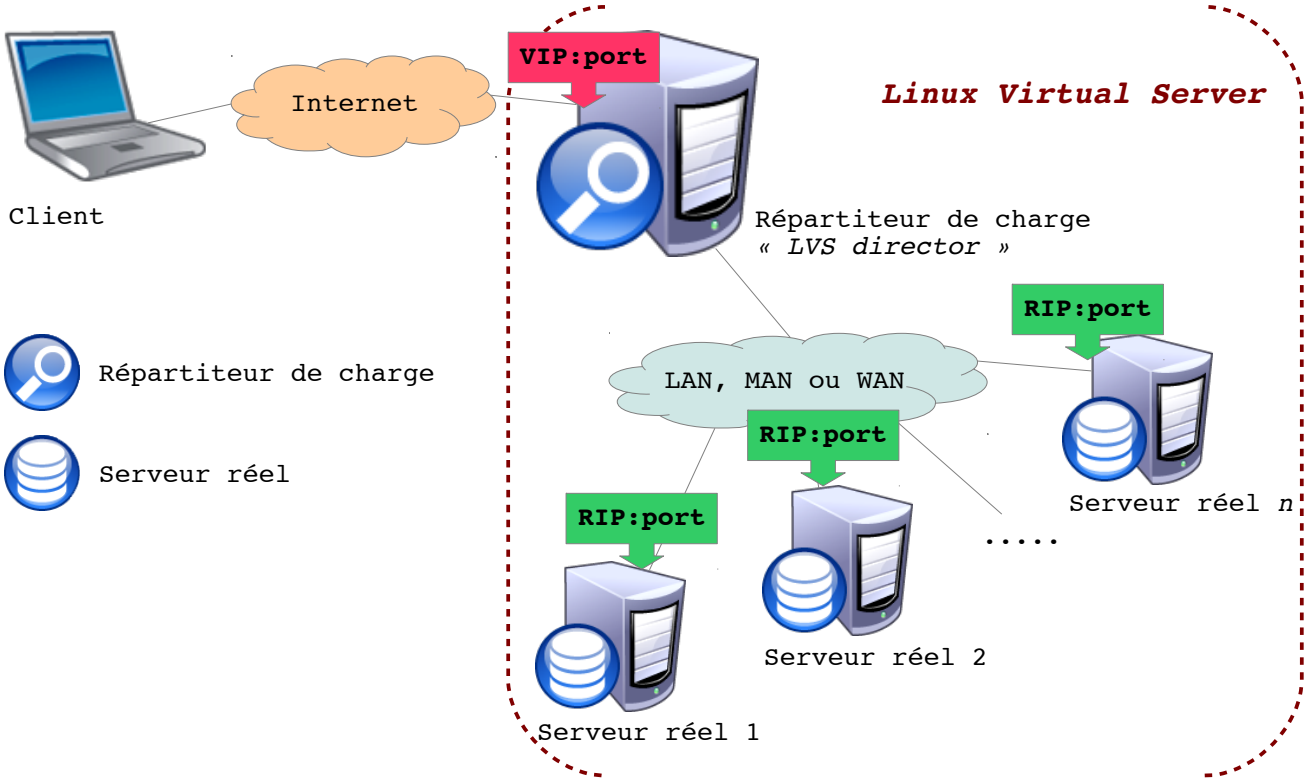
service virtualisé

serveurs redondants



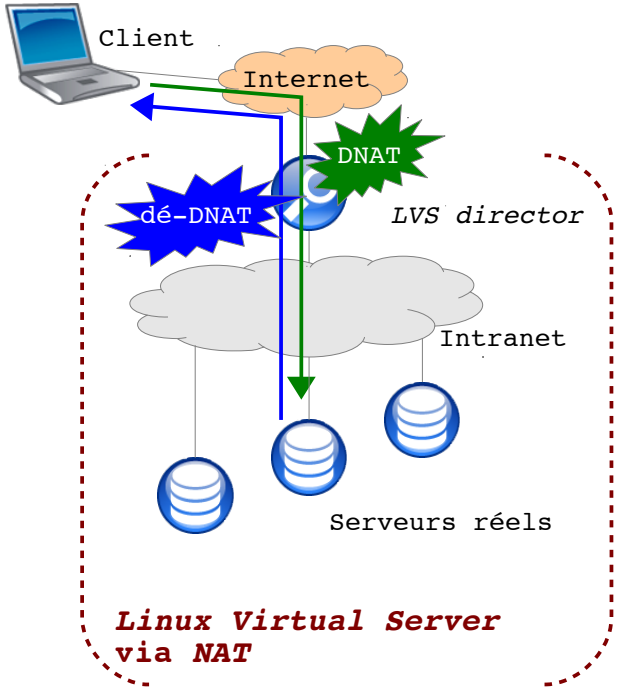
  - problème de *cache* côté client...
  - algorithme de répartition unique (tourniquet)...
  - *TTL* (*Time To Live*) et réactivité en cas de panne...

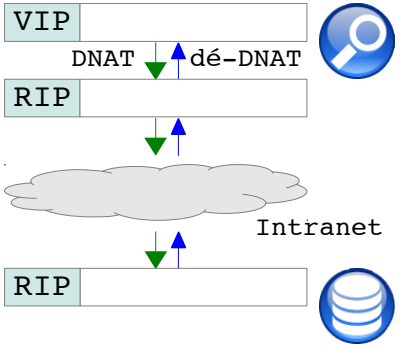
- Virtualisation et répartition de charge à partir d'un élément actif



C'est bon chef ! Il suffit de virtualiser sur un virtualisateur et de répartir sur un répartiteur.

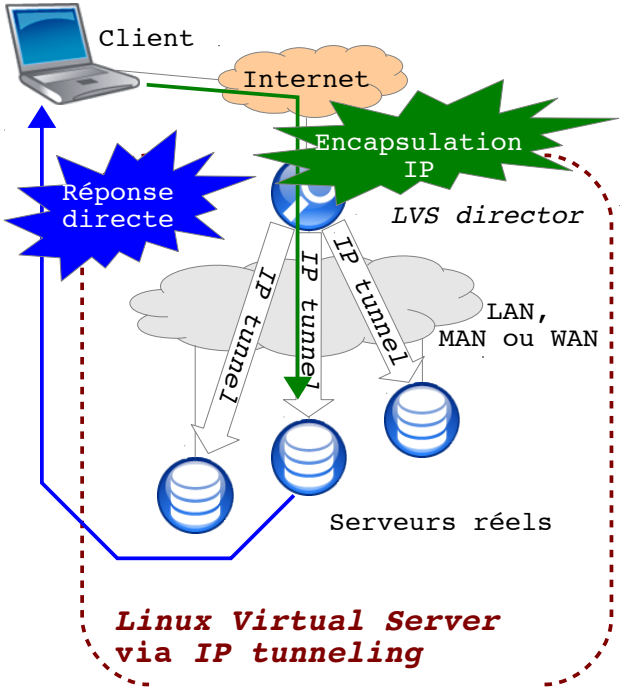
- Mode de redirection des paquets : *NAT*

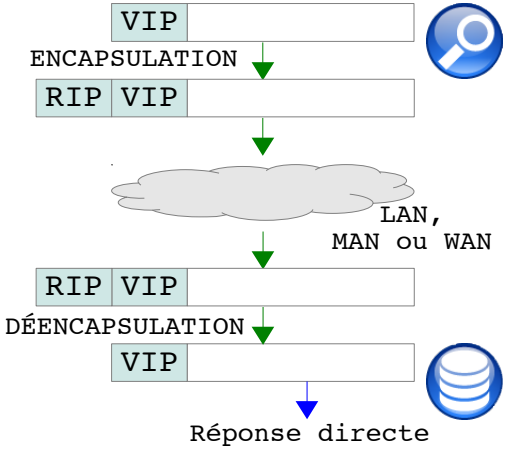




- déploiement uniquement sur réseaux « maitrisés »
- serveurs réels adressés en public ou privé
- répartiteur = passerelle par défaut des serveurs réels
  - nécessiter de maitriser le routage dans *l'intranet*
  - passage « obligé » des paquets retour : attention aux performances du *NAT*...

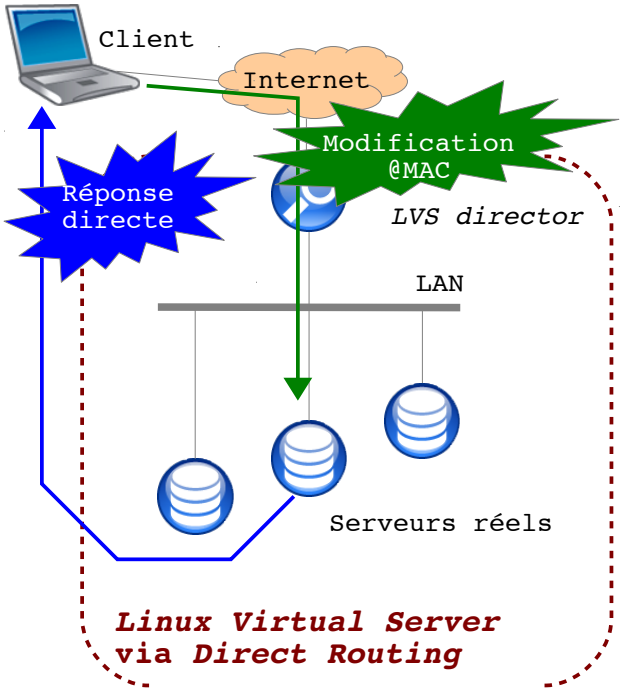
- Mode de redirection des paquets : *IP Tunneling*

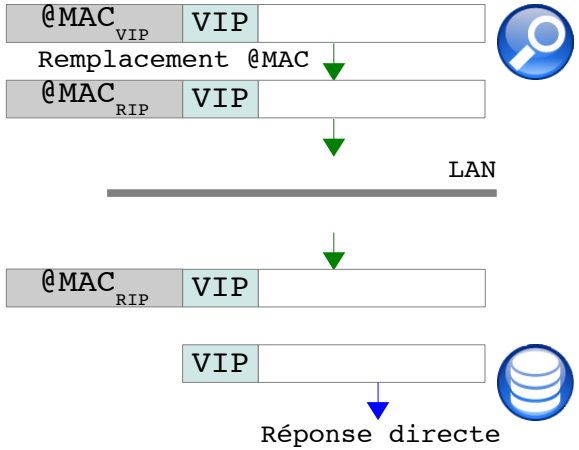




- déploiement possible sur réseaux étendus et non « maitrisés »
- serveurs réels adressés en public
- routage asymétrique : réponse directe au client performante
- tunnels *IP* :
  - construction manuelle et difficile
  - performance de l'encapsulation *IP* ?

- Mode de redirection des paquets : *Direct routing*





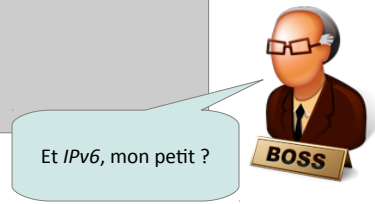
- déploiement uniquement sur réseau local
- serveurs réels adressés en *public*
- routage asymétrique : réponse directe au client performante
- manipulation *@MAC* simple et rapide : bonnes performances entre le répartiteur et les serveurs réels

- Choix du mode de redirection
  - en fonction des contraintes/possibilités du réseau
  - en fonction du niveau de performance souhaité
- Algorithmes de répartition de charge
  - configuration *IPVS* : service virtuel → réparition(serv1, serv2, serv3, …)
  - 10 algorithmes de répartition possible :
    - *Round-Robin* (**rr**), *Weighted Round-Robin* (**wrr**)
    - *Least-Connection* (**lc**), *Weighted Least-Connection* (**wlc**), *Locality-Based Least-Connection* (**lblc**), *Locality- Based Least-Connection with Replication* (**lblcr**)
    - *Destination Hashing* (**dh**), *Source Hashing* (**sh**)
    - *Shortest Expected Delay* (**sed**)
    - *Never Queue* (**nq**)
  - plus de détails sur : *http://www.linuxvirtualserver.org/docs/scheduling.html*

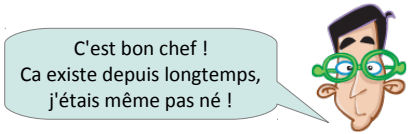
- Configuration *IPVS* avec la commande `ipvsadm`

```
$ ipvsadm --add-service --tcp-service reseau.ciril.fr:80 --scheduler rr

$ ipvsadm --add-server  --tcp-service reseau.ciril.fr:80 --real-server tic.ciril.fr:80 --gatewaying
$ ipvsadm --add-server  --tcp-service reseau.ciril.fr:80 --real-server tac.ciril.fr:80 --gatewaying
```

```
$ ipvsadm  --list

Prot LocalAddress:Port   Scheduler Flags
TCP  reseau.ciril.fr:80  rr
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
  -> tic.ciril.fr:80              Route   1      0          3288
  -> tac.ciril.fr:80              Route   1      0          3298
```



- *IPVS vs. IPv6*
  - *Linux* n'a pas attendu la fin des adresses *IPv4* !
  - depuis 2008 le noyau 2.6.28-rc3 et *ipvsadm* 1.25 sont « *IPv6 aware* »



The end

Starring

…… as The Boss

…… as The Administrator

~~The end~~

To be continued

- *IPVS* = « virtualisation et répartition *1.0* »
  - problématiques non traitées :
    - panne du répartiteur ?
    - prise en compte des indisponibilités des serveurs réels : pilotage dynamique *d'IPVS* ?
    - comment industrialiser le tout ?

- Passons à la « virtualisation et répartition *2.0* »
  - laisser *IPVS* faire son *job*
  - utiliser des outils tiers pour le reste
    - *Piranah, suralived, Linux-HA*
    - ***Keepalived***

Au programme

- Contexte
- *Linux* et *Netfilter*
- *IPVS*
- *Keepalived*
- La quadrature du répartiteur
- Exemples
- Conclusion

- Il était une fois, en l'an 2000, …
  - *Alexandre CASSEN* recherche comment piloter et compléter *IPVS*...
    … résultat : il n'existe pas grand chose !
  - Il décide de développer *Keepalived* selon les principes :
    - utiliser *IPVS* pour la gestion du service virtualisé et sa répartition de charge
    - développer un module de test / vérification des serveurs réels : *HEALTHCHECKERS*
    - développer un module de fiabilisation et résistance aux pannes d'une adresse *IP* : *VRRP*

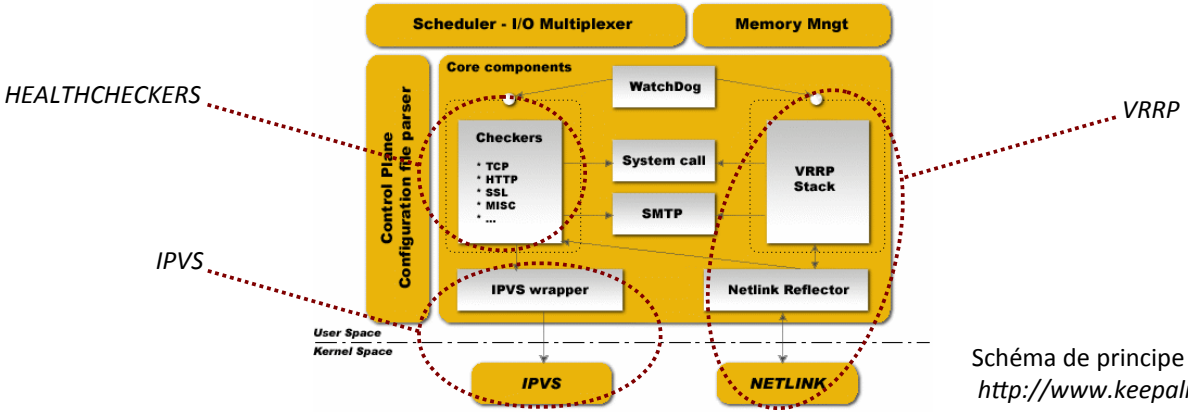
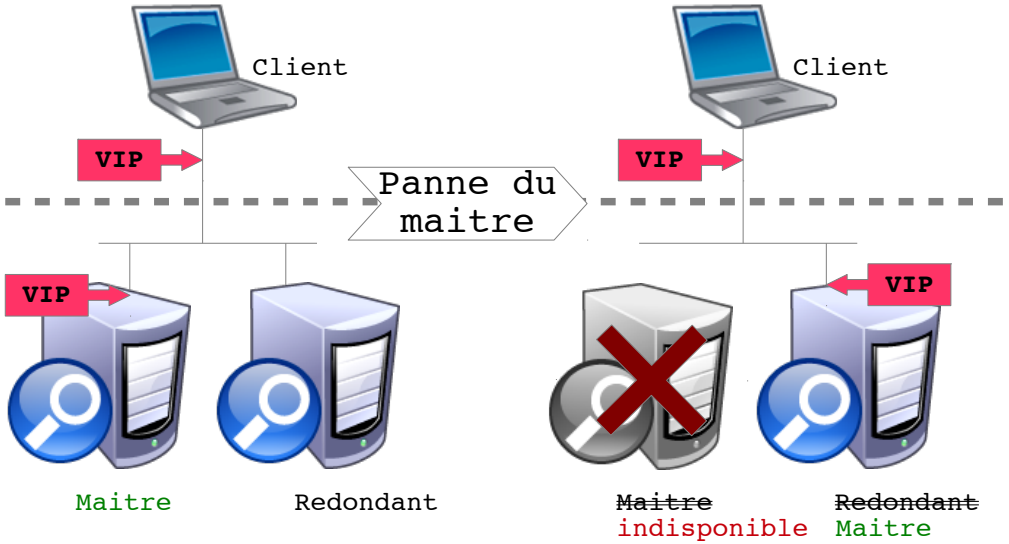


Schéma de principe *Keepalived*
http://www.keepalived.org/software_design.html

- *VRRP : failover IP*
  - fiabiliser un service = redonder le service sur plusieurs serveurs... ça vaut aussi pour le répartiteur !



Keepalived - VRRP

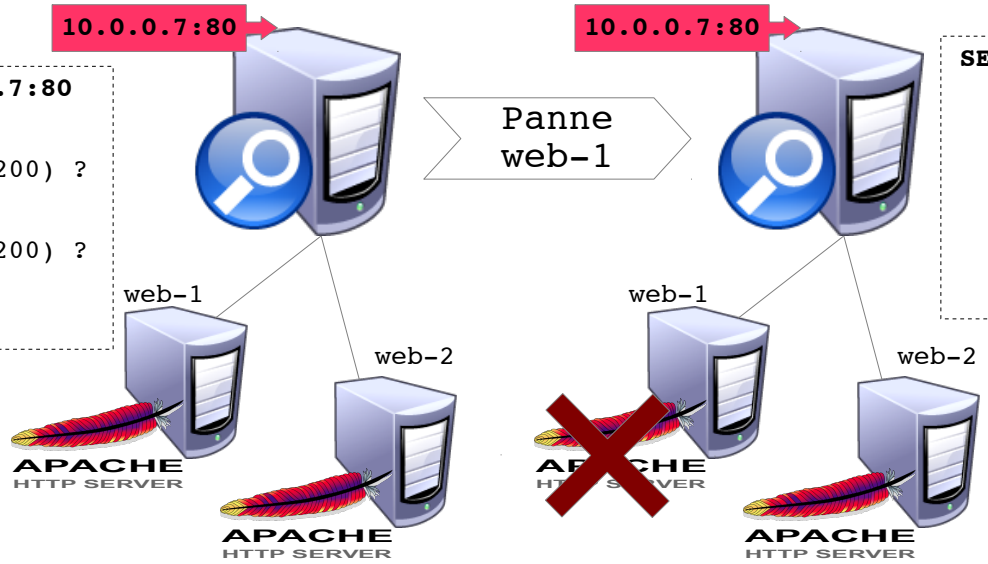
- *VRRP : failover IP*
  - *Virtual Router Redundancy Protocol*
  - protocole standardisé par les *RFC3768* et *RFC579*
  - échanges *multicast* (224.0.0.18) entre les nœuds *VRRP*
  - *« Gratuitous ARP »* pour forcer l'apprentissage du nouveau maître

- *Keepalived* implémente :
  - le protocole *VRRP IPv4*
  - le protocole *VRRP IPv6* (mai 2010 version 1.2.0)
  - des « facilités » d'interaction avec les changements d'états *VRRP*
    - envoi de *mails*, exécution de *scripts*, ...

Keepalived - HEALTHCHECKERS

- *IPVS* + *HEALTHCHECKERS* : service réseau « vraiment » fiabilisé
  - « Mes serveurs réels sont-ils opérationnels ? »
  - « Mon répartiteur connait-il les serveurs réels opérationnels ? »
  - idée : rendre la configuration *IPVS* dynamique

- *IPVS* + *HEALTHCHECKERS* : service réseau « vraiment » fiabilisé



SERVICE VIRTUEL : 10.0.0.7:80

→ web-1:80
GET / web-1 == HTTP(200) ?

→ web-2:80
GET / web-2 == HTTP(200) ?

SERVICE VIRTUEL : 10.0.0.7:80

→ web-1:80
GET / web-1 != HTTP(200)

→ web-2:80
GET / web-2 == HTTP(200) ?

ETAT *IPVS* RÉPARTITEUR

10.0.0.7:80
→ web-1:80
→ web-2:80

Panne
web-1

ETAT *IPVS* RÉPARTITEUR

10.0.0.7:80
→ web-2:80

Keepalived - HEALTHCHECKERS

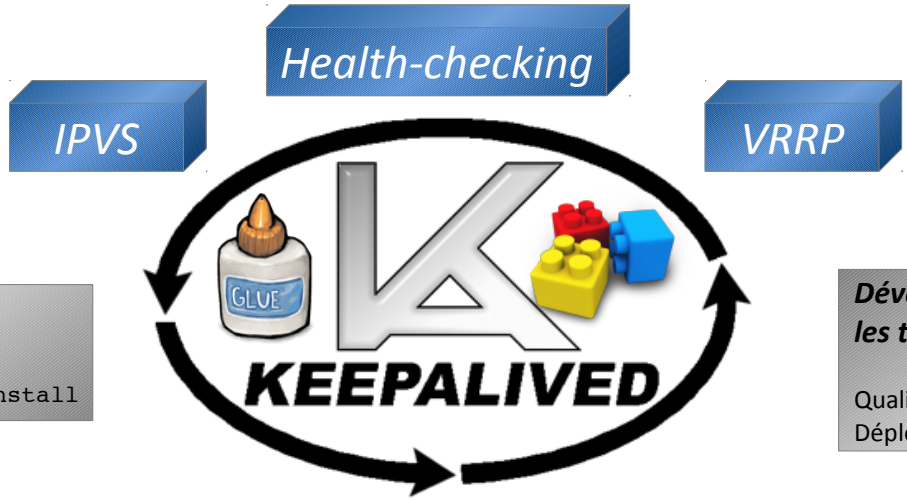
- *IPVS + HEALTHCHECKERS - Keepalived* implémente :
  - la définition des services à virtualiser (via *IPVS*) et
    la définition des serveurs réels, la redirection des paquets et la répartition de charge (via *IPVS*)
    - en *IPv4*
    - en *IPV6* depuis sa version 1.2.2 (janvier 2011)
  - des tests « standards » embarqués
    - *HTTP_GET*, *SSL_GET*, *SMTP_CHECK*, *TCP_CHECK*
  - des possibilités de tests « maison »
    - via la procédure *MISC_CHECK*
  - des « facilités » d'interaction avec les changements d'états des serveurs
    - exécution de *scripts,* changement de poids pour les algorithmes pondérés, ...

Clic-clac, la haute disponibilité est dans le sac !

- *Keepalived* n'est pas magique, juste logique...
  - au départ il y avait :

Clic-clac, la haute disponibilité est dans le sac !

- *Keepalived* n'est pas magique, juste logique...
  - ... et puis ...



Installation facile

```
apt-get install … ou rpm -i …
ou ./configure ; make ; make install
```

Développement de qualité et suivant les technologies émergentes

Qualité du code *C*, support *IPv6*, ...
Déploiements sur des infrastructures majeures

Configuration compréhensible et syntaxe élémentaire

```
/etc/keepalived/keepalived.conf
```

Fonctionnalités « prêtes à l'usage » et extensibles

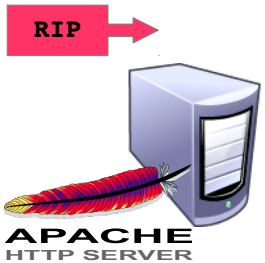
Facilités *VRRP* et *HEALTHCHECKERS*
Tests « standards » et *MISC_CHECK*

Au programme

- Contexte
- *Linux* et *Netfilter*
- *IPVS*
- *Keepalived*
- La quadrature du répartiteur
- Exemples
- Conclusion

La quadrature du répartiteur

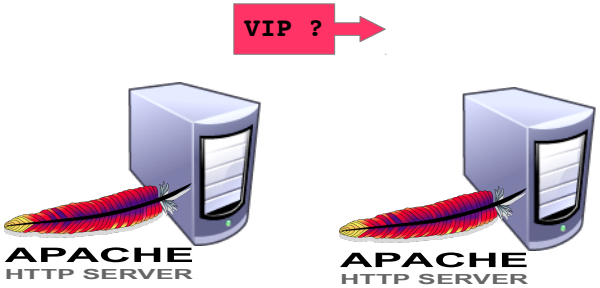
- La quadrature du répartiteur : { 1 + 1 + 1 + 1 = 2 }
  - une idée « originale » de l'équipe réseau du CIRIL



{ 1 = 1 }

La quadrature du répartiteur

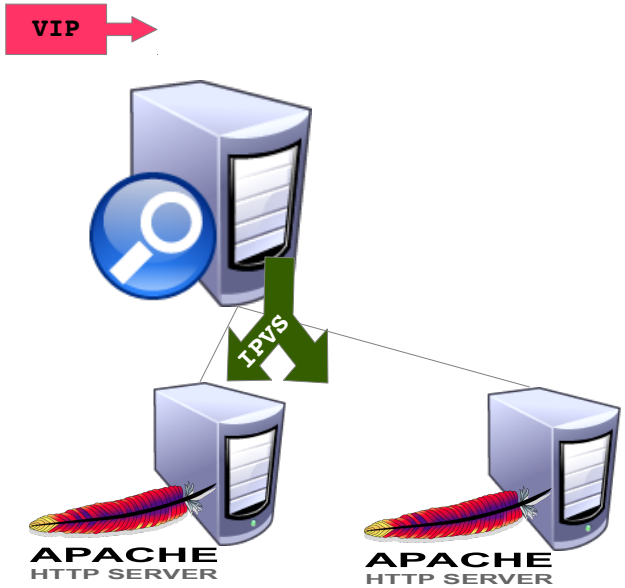
- La quadrature du répartiteur : { 1 + 1 + 1 + 1 = 2 }
  - une idée « originale » de l'équipe réseau du CIRIL



{ 1 + 1 = 2 }

La quadrature du répartiteur

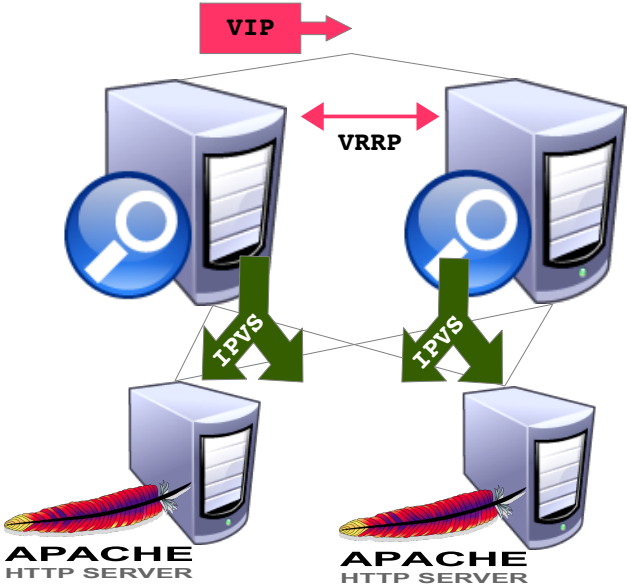
- La quadrature du répartiteur : { 1 + 1 + 1 + 1 = 2 }
  - une idée « originale » de l'équipe réseau du CIRIL



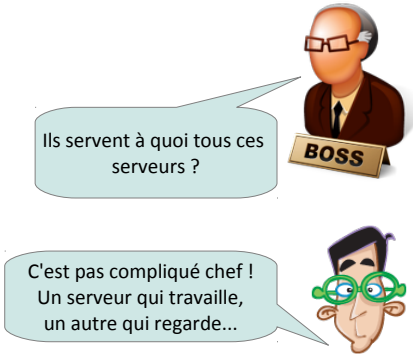
{ 1 + 1 + 1 = 3 }

La quadrature du répartiteur

- La quadrature du répartiteur : { 1 + 1 + 1 + 1 = 2 }
  - une idée « originale » de l'équipe réseau du CIRIL

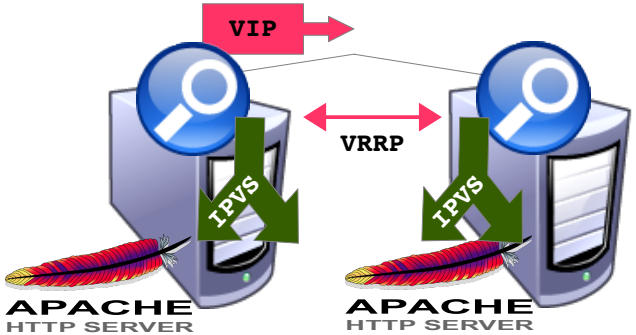


{ 1 + 1 + 1 + 1 = 4 }



La quadrature du répartiteur

- La quadrature du répartiteur : { 1 + 1 + 1 + 1 = 2 }
  - une idée « originale » de l'équipe réseau du CIRIL



- Fusion « service applicatif » et « répartiteur de charge »
- Economie de 2 serveurs (sur 4) !
- Autres avantages :
  - fonctionnement du service en mode « actif-actif »
  - fonction de répartition dédiée au service applicatif
    - configuration et fonctionnement simplifiés
    - performances dédiées au service
  - modèle extensible par « simple » duplication

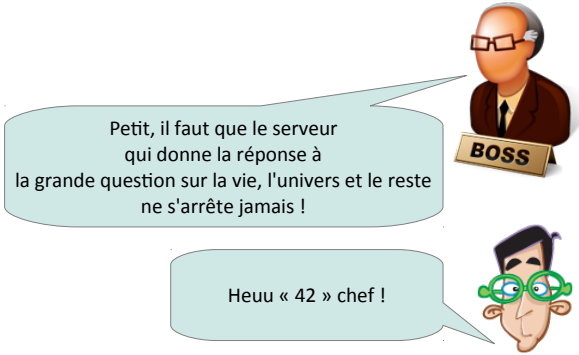
{ 1 + 1 + 1 + 1 = 2 }

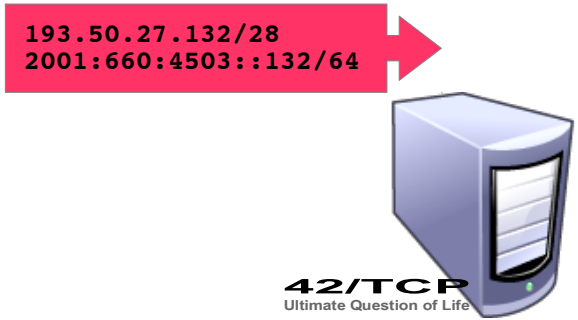
Au programme

- Contexte
- *Linux* et *Netfilter*
- *IPVS*
- *Keepalived*
- La quadrature du répartiteur
- Exemples
- Conclusion

Exemples

- Exemple générique : le service applicatif *42/TCP*

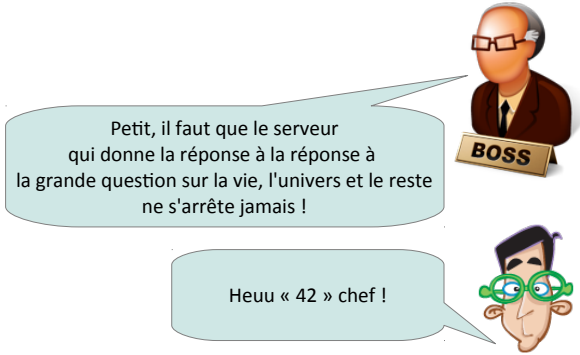


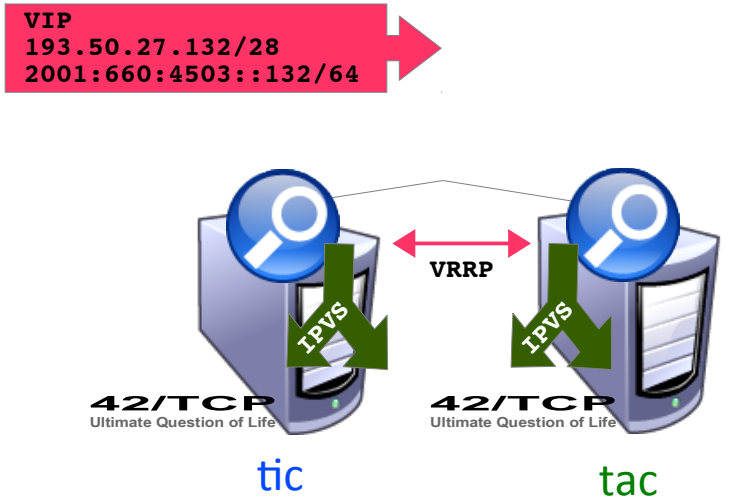


tic

Exemples

- Exemple générique : le service applicatif *42/TCP*

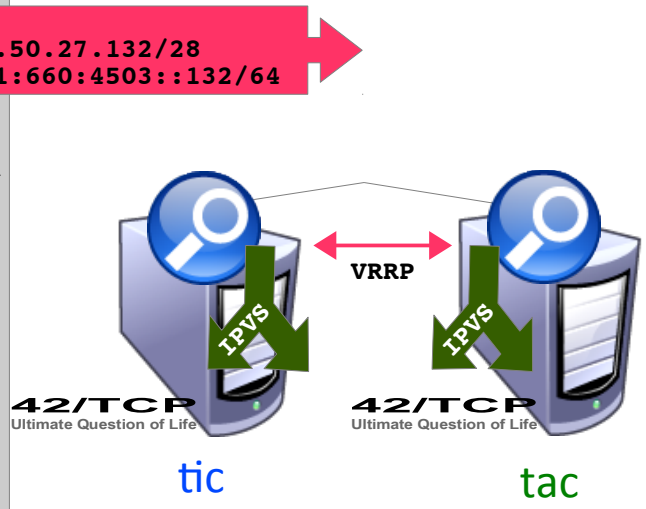




Exemples

- Exemple générique : le service applicatif *42/TCP*

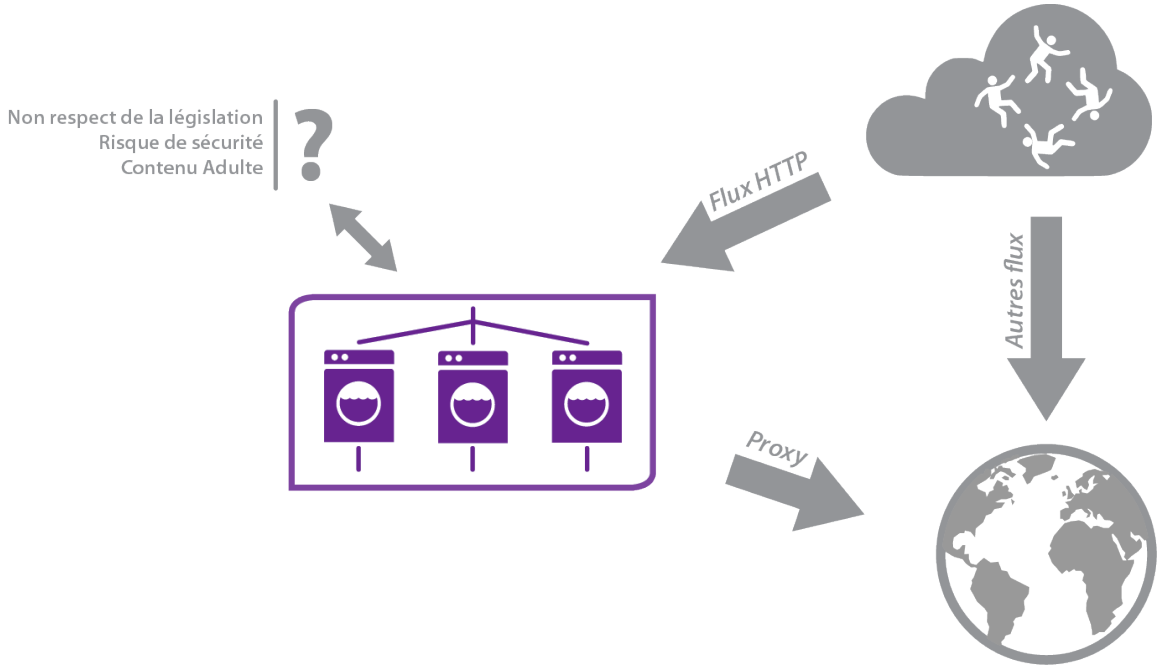
```
# /etc/keepalived/keepalived.conf
vrrp_instance tic {
 virtual_router_id 1
 interface eth0
 priority 200
 virtual_ipaddress {
  193.50.27.132/28
  2001:660:4503::132/64
 }
}

virtual_server 193.50.27.132 42 {
 protocol TCP ; lb_algo rr ; lb_kind DR
 real_server tic 42 {
  TCP_CHECK {
    bindto tic ; connect_port 42
    connect_timeout 3
  }
 }
 real_server tac 42 {
  TCP_CHECK {
    bindto tac; connect_port 42
    connect_timeout 3
  }
 }
}

virtual_server 2001:660:4503::132 42 {
 [ idem ]
}
```




```
# /etc/keepalived/keepalived.conf
vrrp_instance tac {
 virtual_router_id 2
 interface eth0
 priority 100
 virtual_ipaddress {
  193.50.27.132/28
  2001:660:4503::132/64
 }
}
```

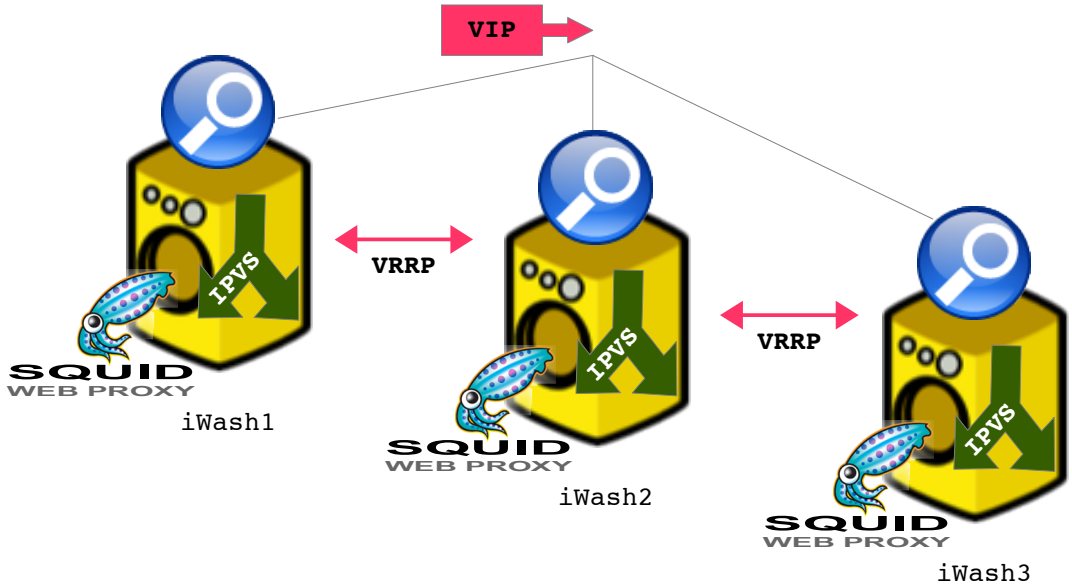
Exemples

- Exemple service de filtrage web : *iWash* (*cf.* poster JRES'2011)



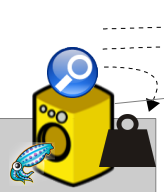
Exemples

- Exemple service de filtrage web : *iWash*



Exemples

- Exemple service de filtrage web : *iWash*




```
vrrp_instance iwash1 {
 priority 150
 virtual_ipaddress {
  ...
 }
}

virtual_server 10.10.10.10 3128 {
 protocol TCP ; lb_algo wrr ; lb_kind DR

 real_server iwash1 3128 {
  MISC_CHECK {
   misc_path "/opt/bin/iweight -p iwash1"
   misc_dynamic
  }
 }

 real_server iwash2 3128 {
  MISC_CHECK {
   misc_path "/opt/bin/iweight -p iwash2"
   misc_dynamic
  }
 }

 real_server iwash3 3128 {
  MISC_CHECK {
   misc_path "/opt/bin/iweight -p iwash3"
   misc_dynamic
  }
 }
}
```

```
vrrp_instance iwash2 {
 priority 100
 virtual_ipaddress {
  ...
 }
}

virtual_server 10.10.10.10 3128 {
 protocol TCP ; lb_algo wrr ; lb_kind DR

 real_server iwash1 3128 {
  MISC_CHECK {
   misc_path "/opt/bin/iweight -p iwash1"
   misc_dynamic
  }
 }

 real_server iwash2 3128 {
  MISC_CHECK {
   misc_path "/opt/bin/iweight -p iwash2"
   misc_dynamic
  }
 }

 real_server iwash3 3128 {
  MISC_CHECK {
   misc_path "/opt/bin/iweight -p iwash3"
   misc_dynamic
  }
 }
}
```

```
vrrp_instance iwash3 {
 priority 50
 virtual_ipaddress {
  ...
 }
}

virtual_server 10.10.10.10 3128 {
 protocol TCP ; lb_algo wrr ; lb_kind DR

 real_server iwash1 3128 {
  MISC_CHECK {
   misc_path "/opt/bin/iweight -p iwash1"
   misc_dynamic
  }
 }

 real_server iwash2 3128 {
  MISC_CHECK {
   misc_path "/opt/bin/iweight -p iwash2"
   misc_dynamic
  }
 }

 real_server iwash3 3128 {
  MISC_CHECK {
   misc_path "/opt/bin/iweight -p iwash3"
   misc_dynamic
  }
 }
}
```

Exemples

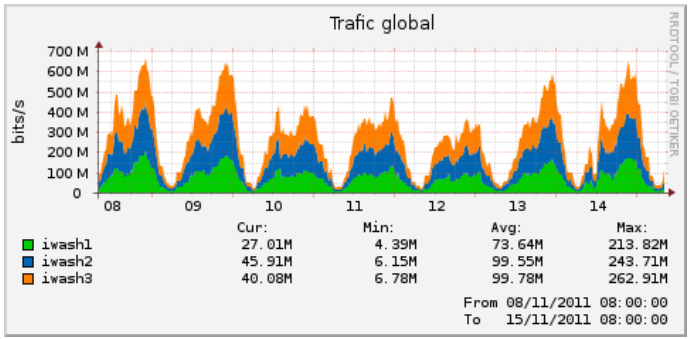
- Exemple service de filtrage web : *iWash*

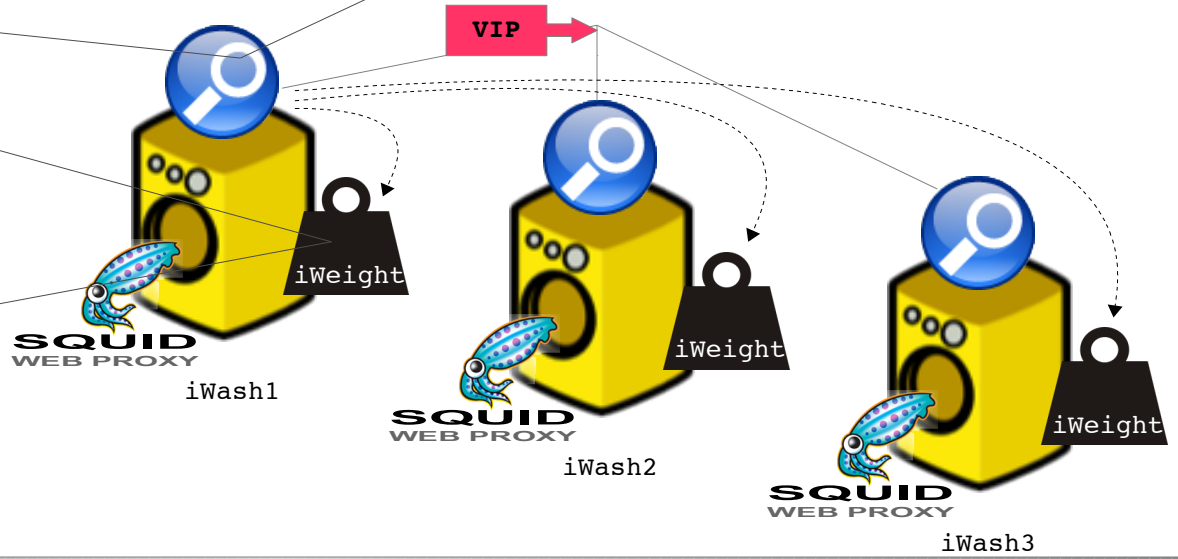
```
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  10.10.10.10:3128 wrr
  ->      iwash1:3128             Local   10     0          10720
  ->      iwash2:3128             Route   20     0          14794
  ->      iwash3:3128             Route   20     0          14780
```

Poids dynamiques

```
weight  =   ManualWeight
          + SquidRunning
          + RedirectToCorpURIRunning
          + OlfeoRunning
          + SquidLoad
          + OlfeoLoad
          + ...
```

weight = capacité de traitement





iWeight Développement « maison » CIRIL

Au programme

- Contexte
- *Linux* et *Netfilter*
- *IPVS*
- *Keepalived*
- La quadrature du répartiteur
- Exemples
- Conclusion

Conclusion

- *Linux / IPVS / Keepalived*
  - « Le triplet gagnant pour la haute disponibilité ! »
  - Juste un « brossage » du sujet... il faut approfondir

- *Keepalived* : «*THE*» solution de haute disponibilité réseau au CIRIL depuis plus de 7 ans
  - montée en puissance des projets
  - support *IPv4* / *IPv6*
  - systématiquement utilisé pour les nouvelles architectures

- Utilisez, utilisez, utilisez