Optimisation du transport de contenu
2 - Équilibrage de charge local

Christophe Deleuze
Grenoble INP – ESISAR

2018–2019

Introduction

fournisseurs de contenu ont besoin de serveurs :

- puissants
- fiables
- évolutifs

idée

- plusieurs serveurs (partage/redondance)

problème

- interface avec le reste du monde ?
  - service identifié par adr. IP + port

Équilibrage DNS

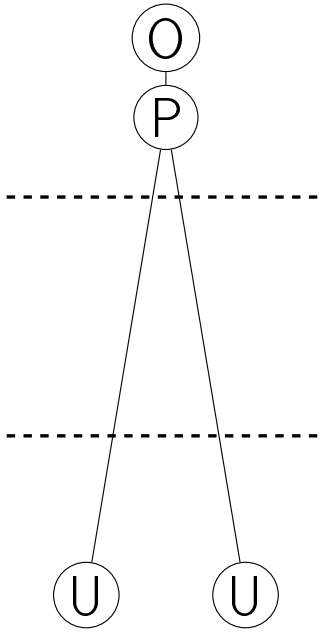
- RFC 1794
- site `www.monserveur.com`
- plusieurs serveurs (réplication)
- tous associés au nom de domaine
- possibilité de changer l'info DNS si un serveur crashe
  - pb : dynamicité (caches DNS)
  - pb : plusieurs adr. IP

Plan

Plan

Reverse proxy

- agit comme un serveur origine
- traduit/relaie les requêtes pour le serveur
- géré par le fournisseur de contenu
  - interface services "legacy" (non HTTP)
  - *performance (accélérateur web)*
  - partitionne/équilibre entre plusieurs serveurs
- N'honore pas les directives `Cache-Control:` du client



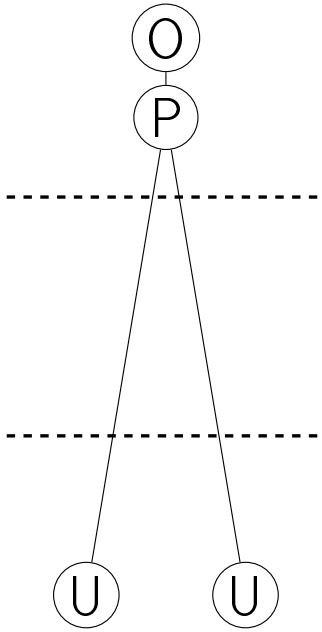
Reverse proxy : performance

- partage dynamique/statique
  - O sert fichiers statiques + pages dynamiques
  - P garde statiques en cache
- cache de pages générées (eg site en PHP)
  - O génère dynamiquement toutes les réponses
  - mais tjs le même résultat !
  - P garde en cache

besoin d'invalidation O $\rightarrow$ P

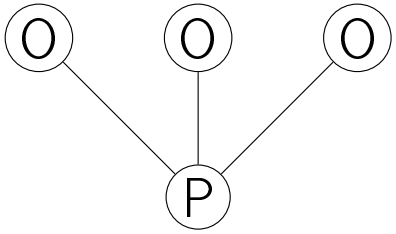
Performance : "spoon feeding"

- P occupé longtemps avec U
  - fenêtre de congestion
  - contrôle de flux (RTT)
- partage du travail
  - O génère
  - P livre



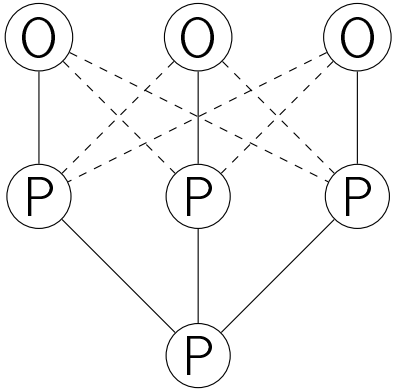
Équilibrage/partition

P peut répartir les requêtes entre plusieurs O

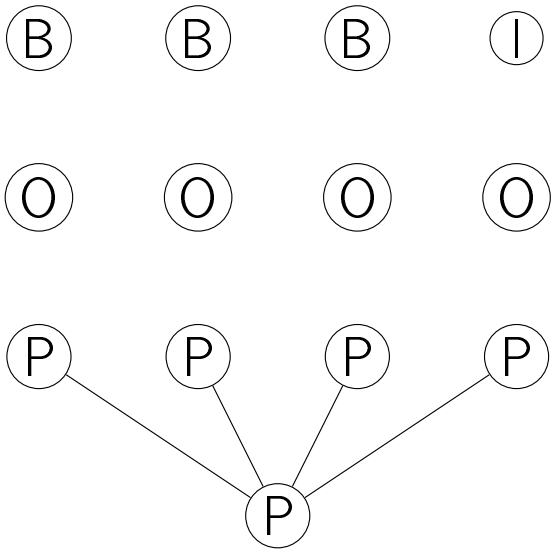


Cluster

plusieurs P cachent et relaient vers plusieurs O



Architecture type wikipedia : un cluster



Plan

Synchronisation

- l'information est dupliquée
- les mécanismes HTTP pour la gestion de la cohérence sont inadaptés
  - besoin d'un "push"
- Deux exemples
  - WCDP (académique)
  - HTCP (utilisé par wikipedia)

Web Content Distribution Protocol – WCDP

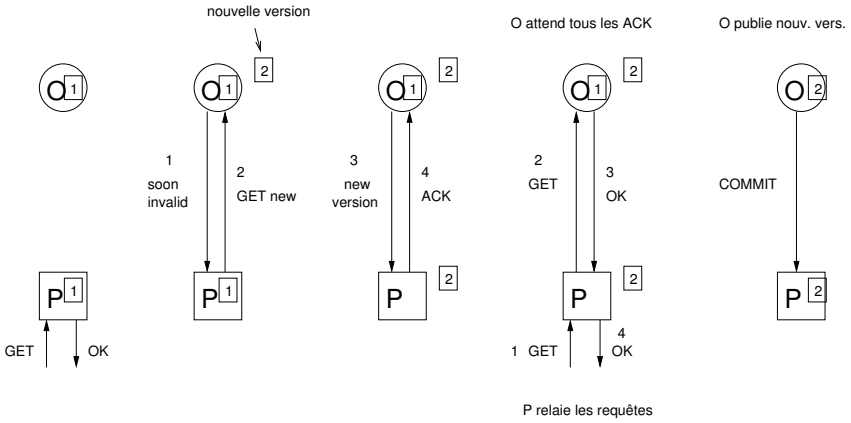
- content group (choix du créateur de contenu)
  - `register <cg>`
- "pseudo-push"
  - invalidation avec rafraich. immédiat
  - ou rafraich. retardé (charge)
- cohérence forte
  - envoie invalidation
  - attend les ACK
  - publie le nouvel objet

WCDP

- cohérence forte avec "push"
  - proxy "pulle" mais relaie les req.
  - qd ts les ACK reçus, origine publie et envoie COMMIT
  - proxy publie à réception du COMMIT
- Δ-cohérence
  - *heartbeat* (revalidation périodique par le serveur)
  - invalidation temporaire

– scalabilité

WCDP – cohérence forte avec push



Hyper Text Caching Protocol - HTCP

rfc2756 HTCP/0.0 - expérimental

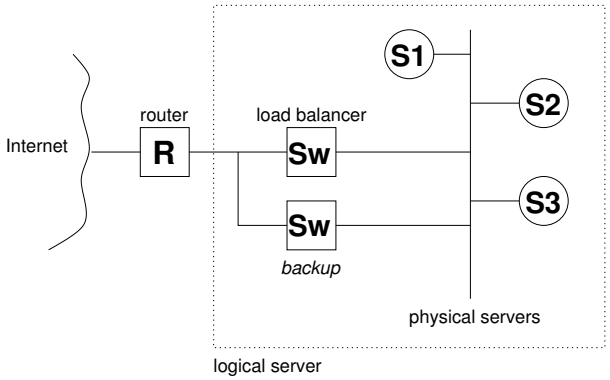
- évolution de ICP (internet cache protocol)
- coopération entre caches
  - requête/réponse optionnelle (sur UDP, ou TCP)
    - `TeST` test de présence en cache : "as-tu ce document ?"
    - `SET` "push" de mise à jour
    - `CLeaR` effacer document
    - `MONitor` informer des modifs du cache
- wikipedia utilise le message CLR
  - message envoyé sur UDP à adr. IP multicast (tous les proxys squid et varnish)
  - pas de réponse/ack

Plan

Ferme de serveurs

- plusieurs serveurs
- une seule adresse IP

- *web switch*
  - répartit
  - surveille
  - backup



Cisco Local Director

The Cisco Local Director Series offers a high-availability, integrated hardware and software solution that intelligently balances the load of user traffic across multiple TCP/IP application servers. Cisco Local Director tracks network sessions and server load conditions in real time, directing each session to the most appropriate server. All physical servers appear as one virtual server, requiring only a single IP address and a single URL for an entire server farm.

...

Cisco Local Director 417G Performance

- *64,000 virtual and real IP addresses*
- *1,000,000 simultaneous TCP connections*
- *400 Mbps throughput*

stoppé en 2003...
remplacé par Content Services Switch (CSS)

Resonate Central Dispatch

Central Dispatch (TM) provides high availability and optimal performance for online services, ensuring a positive end-user experience. Managing all requests at the Web and application tier, ***Resonate Central Dispatch allows multiple Internet servers to act as a single, scalable, reliable, and easily managed server system****. Central Dispatch is the only traffic management product that provisions resources on-the-fly, allowing administrators to bring up and take down servers in real-time, for quick return on investment (ROI).*

...

Load balancing policies ...

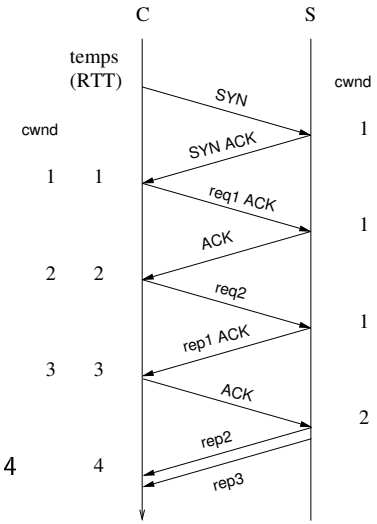
Failover Servers *The failover server feature allows one or more nodes to be specified ...*

Multiple schedulers

SYN Attack Protection *SYN attacks are a type of denial of service (DOS) attack ...*

Rappel : démarrage lent TCP

- fenêtre de congestion `cwnd` (en MSS)
- cwnd++ à chaque ACK de nouvelles données
- ici
  - req. > MSS, sur 2 paquets
  - rép. sur 3 paquets
  - ➠ 4 RTT
- si RTT élevé, peu importe le débit de la ligne
- beaucoup discuté...
  - rfc2581 autorise départ à 2 MSS
  - rfc3390 autorise départ à ≈ 4 ko
  - rfc6928 (exp, google) propose 16 ko



Switch L4

L4 = utilise info couche transport

- reçoit segment `SYN`
- choisit un serveur physique et envoie
- mémorise l'association
- transmet la suite au serveur physique (et retour)
- transmission par :
  - NAT (network address translation)
  - tunnel
  - MAT (MAC address translation)

Switch L4 - NAT

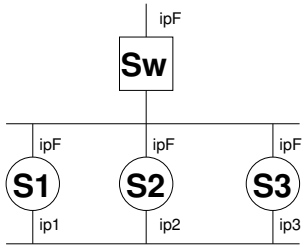
- association con. TCP ↔ adr. IP
- modifie les adresses des paquets
  - source et destination
  - cksum IP et TCP
  - en entrée et en sortie
- topologie quelconque

Switch L4 - tunnel

- *encapsule* le paquet dans un paquet envoyé au serveur physique
- association con. TCP → adr. IP
- réponses directement aux clients
- topologie quelconque
- risque de fragmentation (MTU)

Switch L4 - MAT

- serveurs physiques sur le même LAN
- envoi direct à l'adresse MAC
- association con. TCP → adr MAC
  - tous les serveurs physiques ont la même IP
  - pas de traitement sur trafic en sortie
- souvent réseau de prod/réseau d'administration



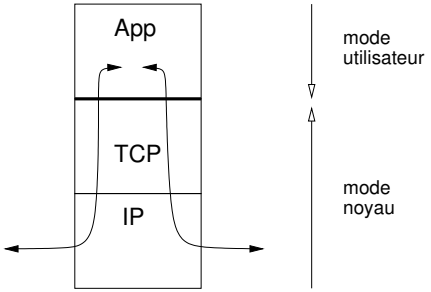
Switch L7

L7 = utilise info couche application

- ouverture de connexion TCP avec le client
- réception (début) requête
- choix serveur physique
- ouvre connexion avec serveur
- relaie (le switch est alors un proxy HTTP... mais qui ne fait ensuite que relayer)

Switch L7 - performance

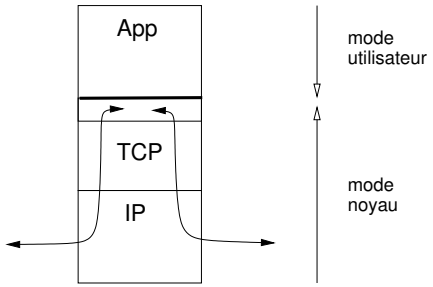
- proxy = processus (espace utilisateur)
- transfert du mode noyau au mode utilisateur et retour
- très coûteux (copies mémoire + changements contexte CPU)



niveau application

Switch L7 - socket splicing

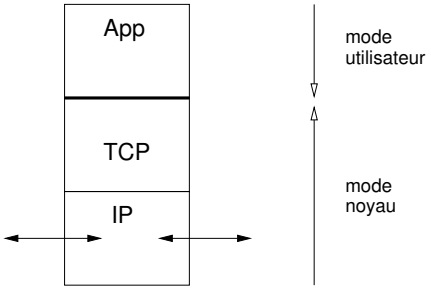
- une fois le serveur physique choisi, le proxy ne fait que relayer...
- passer le relais en mode noyau (aboutement) après envoi de la requête au serveur physique
- Linux
  - `sendfile` (file → socket)
  - `splice` (pipe → socket ou inverse)



niveau socket

Switch L7 - IP splicing

- *fusion* des 2 connexions TCP
- proche du NAT
  - réécritures adresses, ports, numéro de seq.
  - + checksum (optimisation possible)
- pbs
  - options TCP, IPsec
  - très délicat "en cours de route"
  - idem pour désabouter

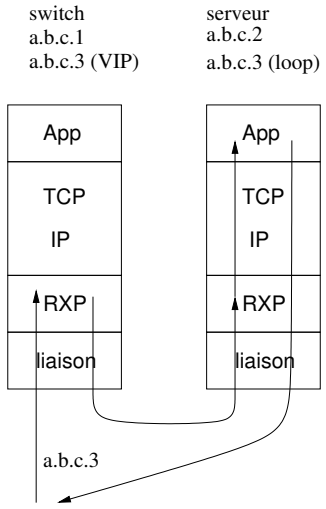


niveau IP

TCP connection hop

© Resonate

1. RXP reçoit, voit dest VIP, répond
2. reçoit requête, choisit serveur `a.b.c.2`
3. encapsule et envoie
4. RXP (serveur) reçoit
5. décapsule et fait remonter
6. serveur répond directement au client



Switch L7 - aboutement – problèmes

- requêtes successives d'un client (avec connexions persistantes)
  - forcer les connexions non persistantes (réponse avec `Connection: close` – beaucoup de serveurs)
  - choisir le serveur physique *par connexion* (et non par requête)
    - mais alors, pourquoi L7 ?
- désabouter à réception requête suivante
  - mais si pipeline ?

Algo de répartition

- but
  - équilibrer la charge
  - partition par type de requêtes/URL
  - persistance
- tourniquet *(round robin)*
- tourniquet pondéré
- moins-de-connexions *(least connections)*
- hachage adresse source
- plus court délai prévu
  - pour chaque serveur C_i nb connex. U_i puissance (cnx/s)
  - délai prévu $\frac{C_i+1}{U_i}$
- par mesure explicite de charge

Surveillance services

les serveurs physiques sont-ils opérationnels ?

- active
  - machine : ping IP
  - service : requête test
  - système : rapports sur la charge CPU/disque/...
- passive
  - observation des réponses aux requêtes en cours

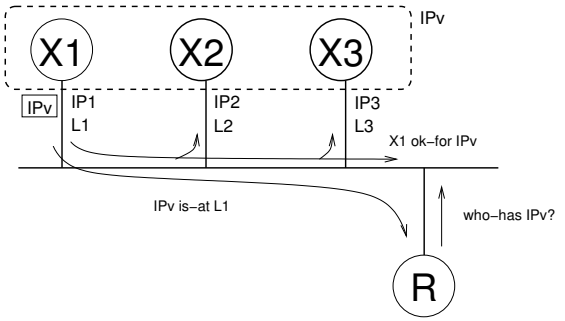
Haute disponibilité

gérer la redondance

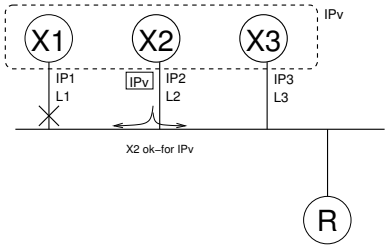
- plusieurs protocoles existants
  - Hot Standby Router Protocol (HSRP) [RFC2281] propriétaire Cisco
  - VRRP Virtual Router Redundancy Protocol (v3) RFC5798
  - CARP Common Address Redundancy Protocol (BSD)
- application courante : redondance d'un routeur d'accès, d'un pare-feu ou d'un serveur
- la même idée générale

Principe général

- chaque X a son adresse IP (IP1, IP2...)
- le X virtuel a l'adr. IPv
- un des X est le maître
  - il "possède" IPv
  - répond aux requêtes ARP pour IPv
  - émet des annonces régulières pour montrer aux autres qu'il est ok



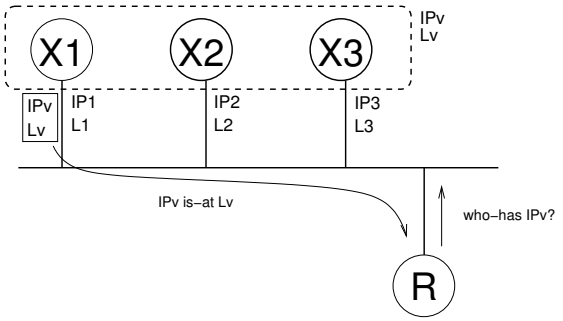
Principe général



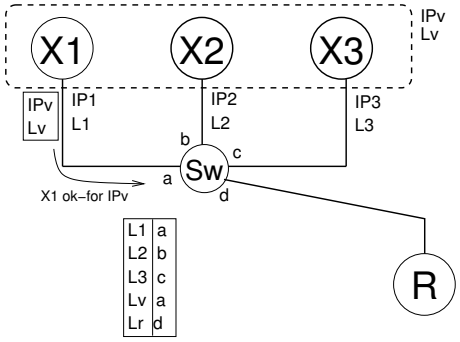
- X1 tombe
- X2 prend IPv
  - répond aux requêtes ARP
  - émet annonces...
- X2 envoie un *gratuitous ARP*
  - annonce diffusée
  - IPv → L2
- pb : `linux: /proc/sys/net/ipv4/conf/eth0/drop_gratuitous_arp` → peut ignorer les annonces ARP !

Adresse liaison virtuelle

- l'adresse IPv est associée à Lv
- Lv "possédée" par la machine active



Adresse liaison virtuelle



- Comment le switch sait-il où est Lv ?
- L'annonce `X1 ok-for IPv` est envoyée depuis Lv !
- Autre solution : jamais de trame partant de Lv (le switch diffuse tjs)

Plan

Slow loris

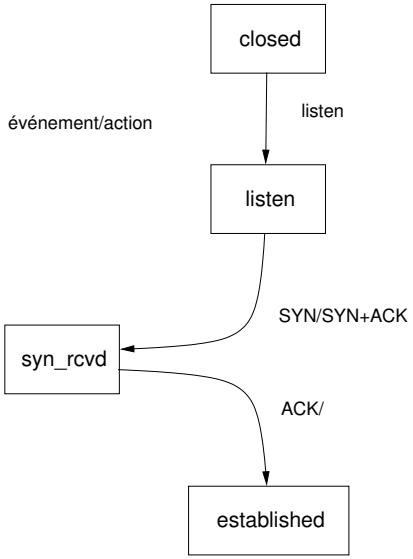
- Un attaquant
  - ouvre de nombreuses connexions TCP
  - écrit très lentement les requêtes
- attaque discrète
  - pas de charge CPU
  - pas de trafic

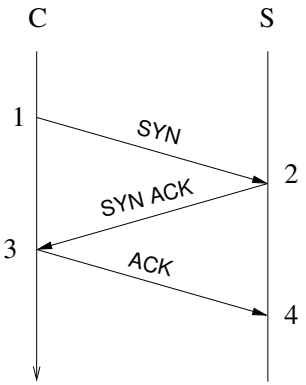
exemple protection (HA-proxy)

```
timeout http-request <timeout>
Set the maximum allowed time to wait for a complete HTTP request
```

l'attaque peut se faire aussi sur la réponse (fenêtre TCP)

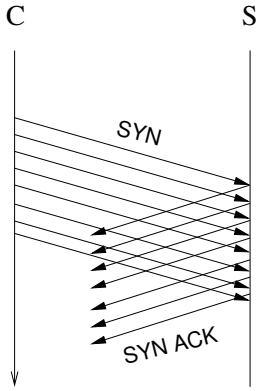
Ouverture de connexion TCP





Déni de service SYN-flooding

- TCB *(transmission control block)* créé à chaque SYN (Linux 1616 octets, FreeBSD 736 octets)
- saturation mémoire...
- ... ou limite au nombre de TCB en état SYN_RCVD
  - nouvelles connexions (légitimes) refusées
- pb : coût envoi SYN/coût allocation TCB



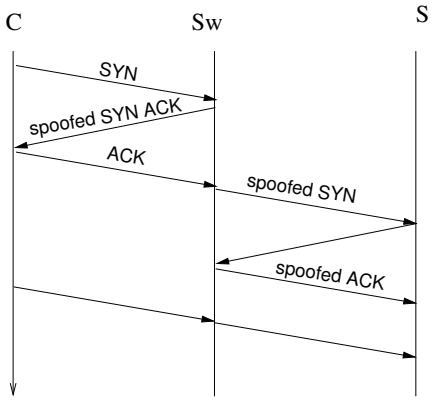
Résistance au SYN flooding

ne pas allouer le TCB complet immédiatement

- "mini" TCB à réception du SYN
  - Linux 96 octets
  - FreeBSD 160 octets
- TCB complet à réception de l'ACK

Protection des serveurs

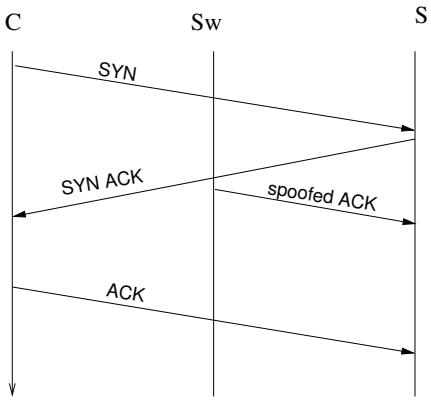
- L7 OK
- L4
  - SYN-ACK spoofing



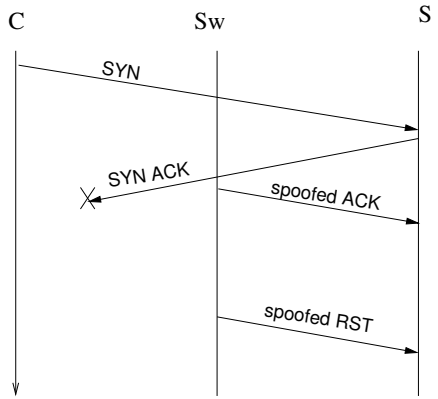
Protection des serveurs

- ACK spoofing

légitime



DOS



SYN cookie (1996)

- SYN cookie : n'alloue rien avant réception de l'ACK
  - encode le TCB (compressé) dans le n° seq
    - 5 bits : t, compteur modulo 32 variant dans le temps
    - 3 bits : encodage du MSS annoncé par le pair
    - 24 bits : fonction secrète des adr. IP, ports et t
  - à réception de l'ACK, décode et crée le TCB
- pb
  - option MSS réduite à 8 valeurs
  - autres options impossibles
  - pas de retransmission du SYN-ACK (viole spec TCP)
- à n'utiliser que quand la file déborde

TCP "moderne"

- option *timestamp*
  - 2×32 bits dans chaque segment
  - 1 valeur locale et un écho
  - calcul plus précis du RTT (RTTM)
  - protection bouclage numéro de séquence (PAWS)
- option *window scale*
  - présente à l'ouverture
  - permet d'appliquer un décalage de n bits à la taille de fenêtre annoncée
- option *SACK (selective acknowledgment)*
  - acquittements non cumulatifs
  - usage négocié par option *sackOK* à l'ouverture

Évolution SYN cookie (2008)

```
→ S 2822954343:2822954343(0) win 5840 <mss 1460,sackOK,
                                      timestamp 1169032 0,nop,wscale 4>
← S 3743796816:3743796816(0) ack 2822954344 win 5672 <mss 1430,sackOK,
                           timestamp 1849894161 1169032,nop,wscale 6>
→ . ack 1 win 365 <nop,nop,timestamp 1169043 1849894161>
→ P 1:102(101) ack 1 win 365 <nop,nop,timestamp 1169043 1849894161>
← . ack 102 win 89 <nop,nop,timestamp 1849894206 1169043>
```

- support de l'option *sackOK*
  - ➠ 1 bit dans le *timestamp*
- support de l'option *window scale*
  - ➠ 8 bits dans le *timestamp*

le premier *timestamp* est imprécis

Linux 2.6.26 net/ipv4/syncookies.c

```
__u32 cookie_init_timestamp(struct request_sock *req)
{
  struct inet_request_sock *ireq;
  u32 ts, ts_now = tcp_time_stamp;
  u32 options = 0;

  ireq = inet_rsk(req);
  if (ireq->wscale_ok) {
    options = ireq->snd_wscale;
    options |= ireq->rcv_wscale << 4;
  }
  options |= ireq->sack_ok << 8;
  ts = ts_now & ~TSMASK;
  ts |= options;
```


En conclusion

- puissance
- haute disponibilité
- évolutivité

problèmes

- synchro interne à la ferme
- résistance aux catastrophes
  - répartition géographique