



**HAPROXY**

Powering Your Uptime

# Les coûts cachés de HTTP/2

Willy Tarreau  
HAProxy Technologies  
FRnOG 26



## Focus

- côté serveur / infrastructure
- on laisse le rendu utilisateur de côté

## HTTP/2 vu du serveur

- 1 connexion par client
- requêtes multiplexées dans des "streams" parallèles et indépendants
- headers compressés donc économie de bande passante
- headers nécessitent une conversion si parsés en HTTP/1, mais peuvent être plus légers si parsés en HTTP/2
- les browsers n'implémentent que la version TLS de HTTP/2 ("h2" et pas "h2c").



# HAProxy

Powering Your Uptime

## Les gains par rapport à HTTP/1.x

- réduction du nombre de connexions concurrentes par client
  - possibilité d'aborder des streams sans fermer la connexion
- => forte réduction du connection rate, les firewalls apprécieront*

## Les pertes

- incitation à maintenir les **connexions idles** plus longtemps  
*=> plus grand nombre de connexions concurrentes au total donc plus grande consommation mémoire et sockets. Ce problème a commencé il y a environ 2 ans avec les browsers effectuant du preconnect en HTTP/1*
- le passage en H2 implique que **tout le contenu** sera délivré ainsi et donc que tout **passera en TLS**, y compris des objets statiques sans intérêt  
*=> les caches de campus/entreprise/ISP deviennent inefficaces, même certains CDN, d'où un accroissement du nombre de requêtes par client sur certains sites*
- des serveurs de statiques auparavant très légers vont devoir supporter des handshakes TLS.
- le **sharding** qui permettait de paralléliser les connexions sur plusieurs noms de hosts va engendrer autant de nouvelles connexions avec leur propre handshake TLS.

## Implications de cette mise en oeuvre forcée de TLS

- côté exploitation : devoir gérer des certificats sur tous les sites
- côté code : devoir implémenter et supporter openssl!  
=> *vraiment pas drôle.*
- côté packaging : intégrer **openssl 1.0.2** (pour support **ALPN**)
- débit réseau : les dernières implémentations de l'AES-NI sont très performantes : **50 Gbps par coeur** en AES128-GCM pour un intel core i7-6700K à 4 GHz  
=> *aucun intérêt à utiliser des accélérateurs crypto pour cette partie.*

## Implications de cette mise en oeuvre forcée de TLS (suite)

- **TLS resume** : pratiquement plus vu qu'on monte une seule connexion.
- **TLS handshake** :
  - **RSA2048** : **2k** sign/s, **72k** verify/s sur le même CPU.
  - **ECDSA256** : **36k** sign/s, **17k** verify/s
  - => note, ce sont les perfs brutes des algos rapportées par openssl, il faut ajouter le "gras" autour comme ECDHE etc...*
- Note: les **CPU end-user** sont intéressants pour la crypto (plus haute fréquence, moins chers, faibles besoins en débit mémoire, mais un seul socket).
- les **cartes crypto** peuvent apporter quelque chose ici, mais ça n'est pas si simple.
- astuce: on peut faire tourner le **code SSL** sur les **mêmes coeurs** que les **drivers réseau** car on n'a jamais les deux en même temps.

## Utilisation d'accélérateurs crypto

- embarquent en général plein de calculateurs parallèles. Ex: une carte donnée pour **35k** RSA2048/s contient **56** processeurs, chacun capable de faire 640 clés/s.
  - => 1: **il faut savoir les occuper**
  - => 2: **ajoutent une forte latence par clé (ex: 1.6 ms contre 600us en soft)**
- les calculateurs sont focalisés sur **certaines algos**. La carte mentionnée ci-dessus affiche des perfs atroces dès qu'on active l'ECDHE (quelques dizaines de clés par seconde). Apparemment elle n'est même pas compatible ECDSA. Des modèles plus récents le seraient.
- beaucoup de **bugs hardware** nécessitant des workarounds softs
- besoin de **drivers** dans le kernel utilisés en userland. **Qui les a audités ?**
- nécessité de **patcher openssl** plus ou moins lourdement, ce qui complique très fortement le maintien à niveau par rapport aux correctifs et oblige dans la pratique à distribuer du code plein de bugs connus.



# HAPROXY

Powering Your Uptime

## Au final, plutôt miser sur les gros CPU ?

- pas si chers que ça :

- 2\*E5-2690v3 => 24 coeurs 2.6 GHz pour moins de \$4.5K

- 2\*E5-2687Wv3 => 20 coeurs à 3.1 Ghz pour le même prix

- 2\*E5-2630v3 => 16 coeurs à 2.4 GHz pour ~\$2K

=> **pour \$10k, 124 GHz sur 2 machines ou 192 GHz (5 machines)**

=> **60 à 100k RSA2048/s pour ce prix, 18 fois plus en ECDSA**

=> **préférer sur des fréquences élevées** pour limiter la latence du handshake.

=> privilégier des machines les **plus génériques** possibles pour favoriser la scalabilité horizontale avec de l'existant.

- flexibles, suivent les **évolutions logicielles** et reçoivent des correctifs

- facile d'**ajouter des machines**, pas besoin de commander du matériel exotique.

- machines reconvertibles même pendant les périodes creuses, (principe du cloud)

## Espoirs

- intel a développé une **API asynchrone** pour openssl, qui a été intégrée en 1.1.0, ça va déjà éliminer le problème du fonctionnement **mono-processus**. Reste à voir si les autres fabricants s'appuieront dessus.

### - ECDSA :

Cf: <http://csrc.nist.gov/groups/ST/ecc-workshop-2015/presentations/session2-andrews-rick.pdf>

- supporté par beaucoup de browsers, dépendent en fait **des root CA** :

- firefox >= 3.6.28

- safari on iOS >= 7 or OSX >= 10.9.1

- MSIE >= Vista / Windows Phone 8

- Chrome & Opera : dépend des OS, s'appuient sur un "trusted root store"

## Espoirs (suite)

### ECDSA (suite)

- 67% des certificats SSL émis le sont par une CA qui sait aussi faire de l'ECDSA  
=> *2 utilisateurs sur 3 devraient pouvoir en obtenir sans changer de fournisseur.*
- OpenSSL 1.0.2 supporte **plusieurs types de certificats** pour un même contexte (SNI), permettant d'activer ECDSA et RSA sur le même domaine
- le prix des processeurs embarqués baisse continuellement et leur puissance augmente. Aujourd'hui pour **\$40** on fait **320** clés RSA2048/s, c'est **10 fois moins** qu'une machine x86 qui coûte **25 fois plus cher**. Le pb reste la latence (12.5 ms). On peut envisager à moyen terme de déléster ces traitements sur des cartes à base de CPU génériques massivement parallèles.