Comparaisons et attaques sur HTTP2

...

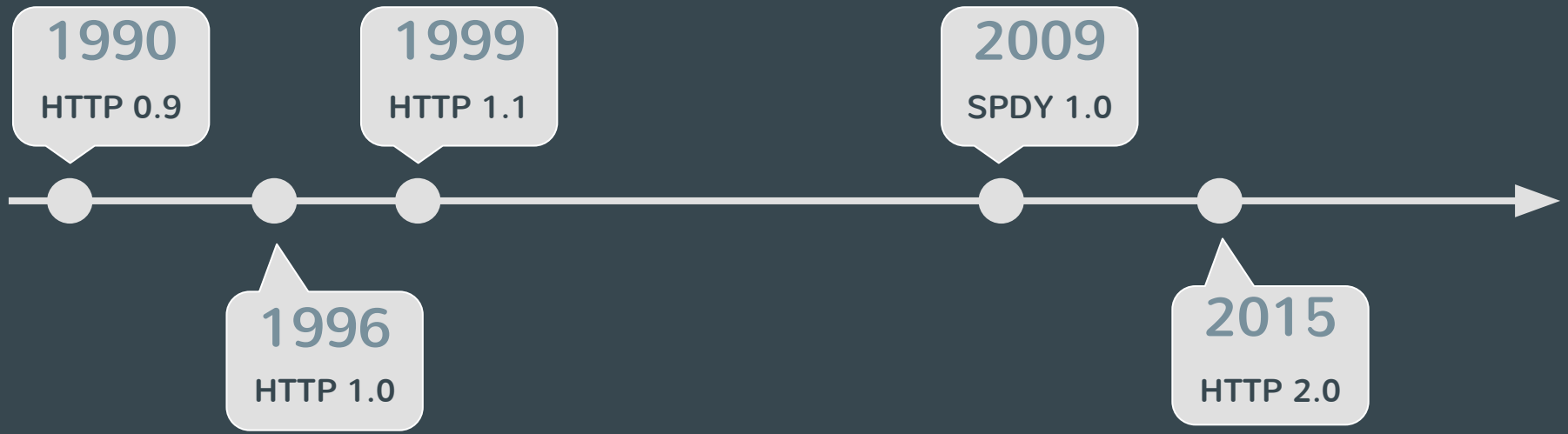
Georges Bossert - 3 juin 2016

@Lapeluche

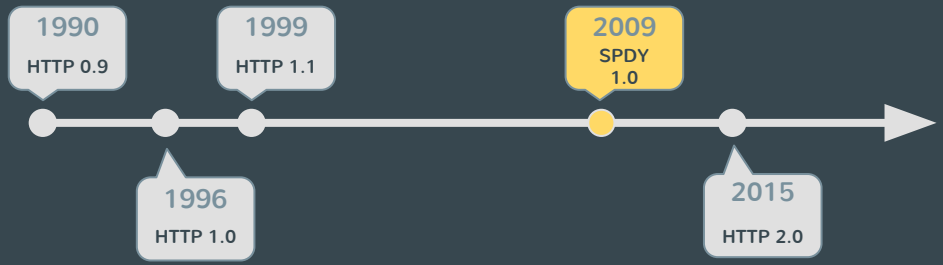
Au programme

1) Présentation **RAPIDE** du protocole HTTP/2
2) Comparaison de piles serveurs
3) Exploitation des résultats

HTTP-Quoi ?


1990
HTTP 0.9
1996
HTTP 1.0
1999
HTTP 1.1
2009
SPDY 1.0
2015
HTTP 2.0

SPDY 1.0



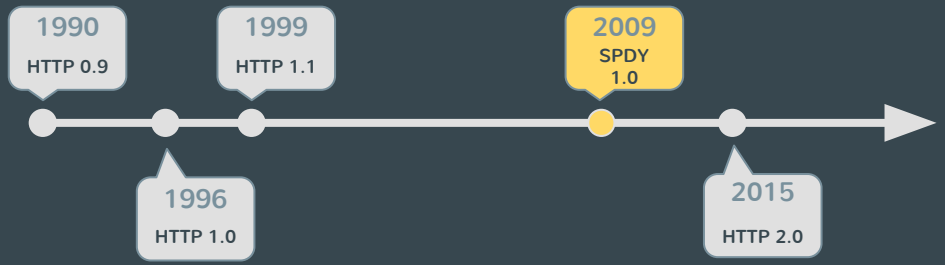
Une amélioration du HTTP proposée par Google

- Objectif principal : Réduire la latence

En HTTP/1.1

- une requête HTTP = une connexion TCP (*head-of-line blocking*)
- le client est TOUJOURS à l'initiative d'un échange de données
- les entêtes ne sont pas compressées (et quelques fois inutiles)
- le contenu des échanges n'est pas toujours compressé

SPDY 1.0



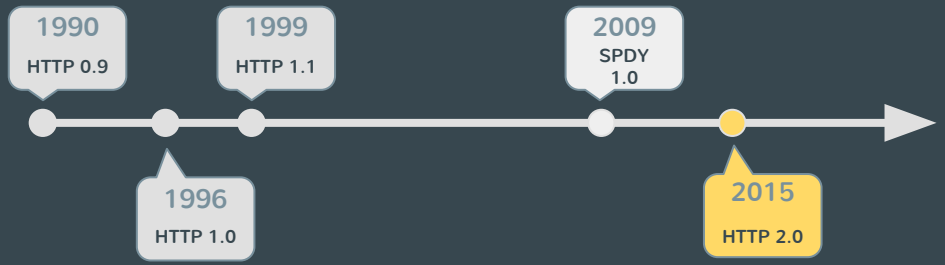
Une amélioration du HTTP proposée par Google

- Objectif principal : Réduire la latence

Principales fonctionnalités

- un nombre illimité de flux concurrents sur une connexion TCP
- priorisation des requêtes
- compression des entêtes
- “Server Push” et “Server Hint”
- SSL Required !

HTTP/2



Standardisation du SPDY

Le premier draft (nov. 2012) : une copie de SPDY

Puis quelques différences:

- **SSL NOT required !**
- Extension TLS : ALPN (et pas NPN)
- HPACK pour la compression (Oracle Attacks: BREACH, CRIME)
- Amélioration de la priorisation,
- (...)

HTTP/2

Protocole binaire et presque tout le temps chiffré

Mise en oeuvre d'une **machine à états** pour le multiplexage de streams

- Un "stream" peut être priorisé, re-priorisé et annulé à tout moment
- Un "stream" peut avoir des dépendances sur d'autres "streams"
- Un "stream" dispose de son propre "control flow"

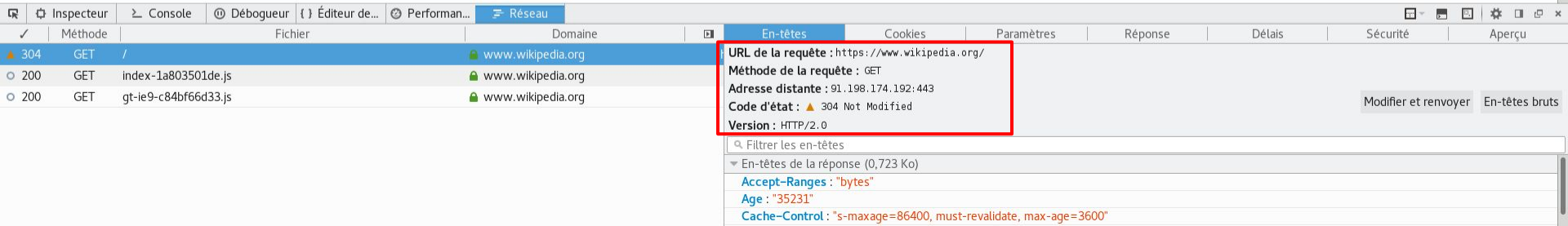
Les entêtes sont compressées

Le serveur peut "pusher" des données aux clients

Emploi de l'Upgrade-mode ou d'une négotiation TLS/ALPN

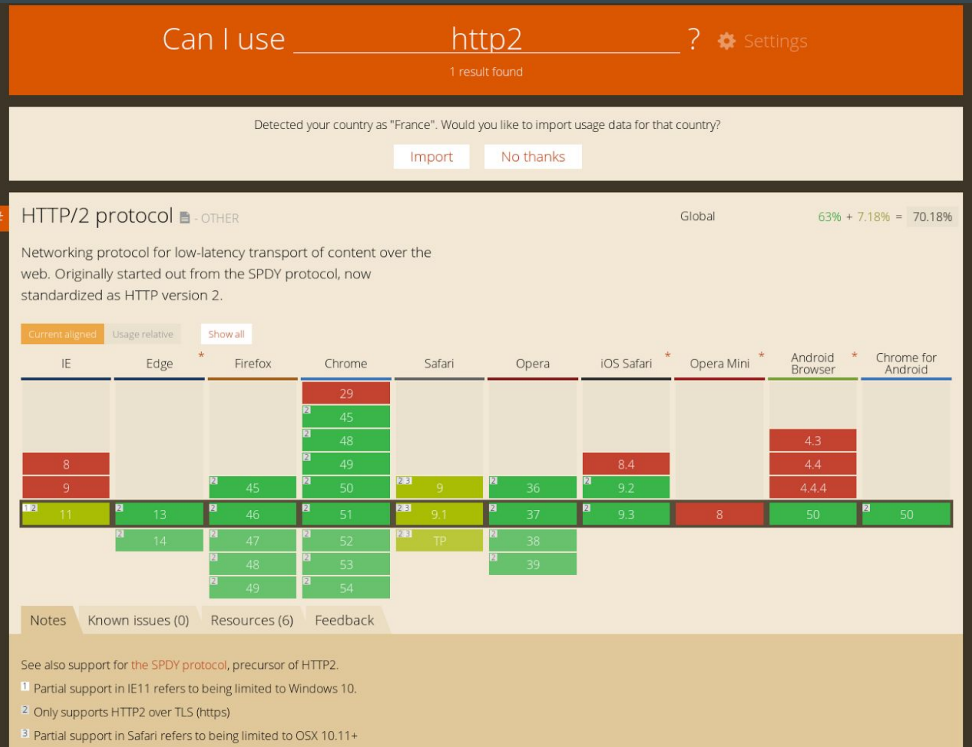
Quelques sites Internet sous HTTP/2

Google, Facebook, Twitter, Wikipedia, Yahoo, Cloudflare, Amazon.com, ...



Une majorité de navigateurs utilise HTTP/2

IE, Edge, Firefox, Chrome, Safari, Opera, IOS Safari, Firefox Android, ...



30/05/2016

Un grand nombre de serveurs compatibles HTTP/2

> 40 implémentations référencées sur

https://github.com/http2/http2-spec/wiki/Implementations

- Aerys,
- Apache Httpd,
- Apache Tomcat,
- Deuterium,
- H2O,
- Jetty,
- Nginx,
- Netty,
- Node-http2,
- OpenLitespeed,
- ...

Implémentations partielles du HTTP/2

- “Push” pas toujours disponible
- “SSL not required” pas toujours respecté

HTTP/2 est complexe

- Protocole très récent
- Nombreux utilisateurs
- Nombreuses implémentations
- Nombreuses fonctionnalités
- Mise en oeuvre de HPACK
- Passage d'un protocole *stateless* à *statefull*
    - Repose sur une machine à états

HTTP/2 est complexe

- Protocole très récent
- Nombreux utilisateurs
- Nombreuses implémentations
- Nombreuses fonctionnalités
- Mise en oeuvre de HPACK
- Passage d'un protocole *stateless* à *statefull*
  - Repose sur une machine à états

Comment aider les développeurs ?

Au programme

1) Présentation rapide du protocole HTTP/2
2) **Comparaison de piles serveurs**
3) Exploitation des résultats

Comparaison de piles protocolaires

- **Piste 1 : Comparer les documentations**
  - Pas toujours à jour
  - Pas suffisamment précises
- **Piste 2 : Comparer les codes sources**
  - Manuellement (très lent) ou Automatiquement (très complexe)
  - Pas toujours facile d'accéder au code source
- **Piste 3 : Collecter puis analyser des traces d'exécutions**
  - Difficulté de mener une comparaison reproductible
  - Complétude de la comparaison limitée au contenu des traces
- **Piste 4 : Inférer l'automate en stimulant l'implémentation**
  - L'implémentation doit être instrumentable
  - La machine à états doit être représentable par un IO-Automata

Comparaison de piles protocolaires

Inférence ACTIVE de l'automate d'une implémentation via LSTAR (L*)

1) Initialisation d'une table d'observation

2) Remplissage de la table d'observation

- Construction d'une séquence de messages et envoi à l'implémentation
- Stockage des réponses dans la table d'observation
- Reset de l'implémentation

3) Transformation de la table d'observation en automate

4) Comparaison de l'automate avec celui de l'implémentation

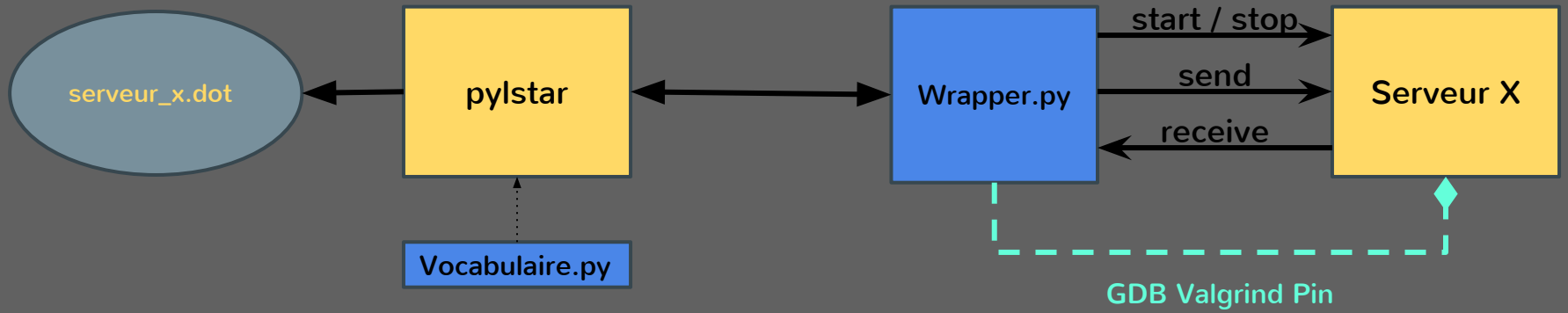
- Génération de séquences aléatoires acceptées par l'automate
- Si une erreur est trouvée, on corrige la table d'observation et on retourne au 2)

5) L'automate inféré est équivalent à celui de l'implémentation

Comparaison de piles protocolaires

Inférence ACTIVE de l'automate d'une implémentation via LSTAR (L*)

- Création d'une implémentation open-source en python : **pylstar**



Comparaison de piles protocolaires

Les serveurs HTTP/2 comparés

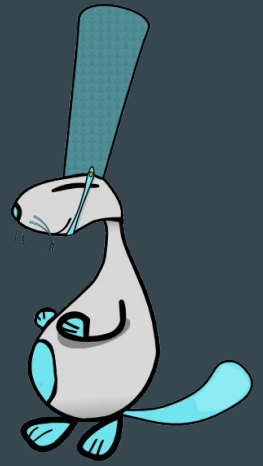
Nom	Info	Version retenue
Apache	Module “mod_http2” (experimental)	2.4.20 (latest) + nghttp2 1.10.0 (latest)
Nginx	http2 pas disponible en stable	1.9.15 (mainline)
H2o	dernière stable (2.0.0 en cours de dev)	1.7.1 (latest release)
Tomcat 9		9.0.0.M4 (latest)

https://github.com/gbossert/http2_compare

- Toutes les sources
- Tous les paramètres de builds
- Toutes les confs

Versions identifiées le 25 avril 2016

Comparaison de piles protocolaires



Construction du vocabulaire d'entrée (vocabulaire.py)

- Lecture des spécifications HTTP/2
- Identification des cas aux limites
  - Négation des MUST, SHOULD, MAY
- Représentation des messages avec Netzob

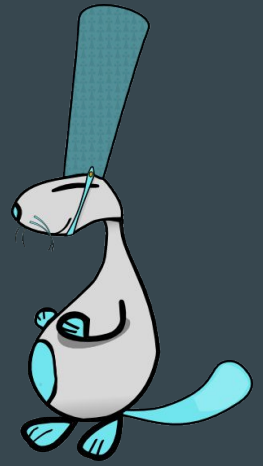
```
638
639 # HTTP/2.0 PING
640 PING = Symbol(name = "Ping")
641 PING.fields = [
642     Field(name="Length"),
643     Field(name="Type", domain=Raw('\x06')),
644     Field(name="Flags", domain=Raw('\x00')),
645     Field(name="Stream Identifier", domain=Raw("\x00\x00\x00\x00")),
646     Field(name="Opaque Data", domain=Raw('\x07\x06\x05\x04\x03\x02\x01\x00')),
647 ]
648 PING.fields[0].domain = Size(PING.fields[4:],
649                              dataType = Raw(nbBytes=3, unitSize=AbstractType.UNITSIZE_32))
```

Comparaison de piles protocolaires

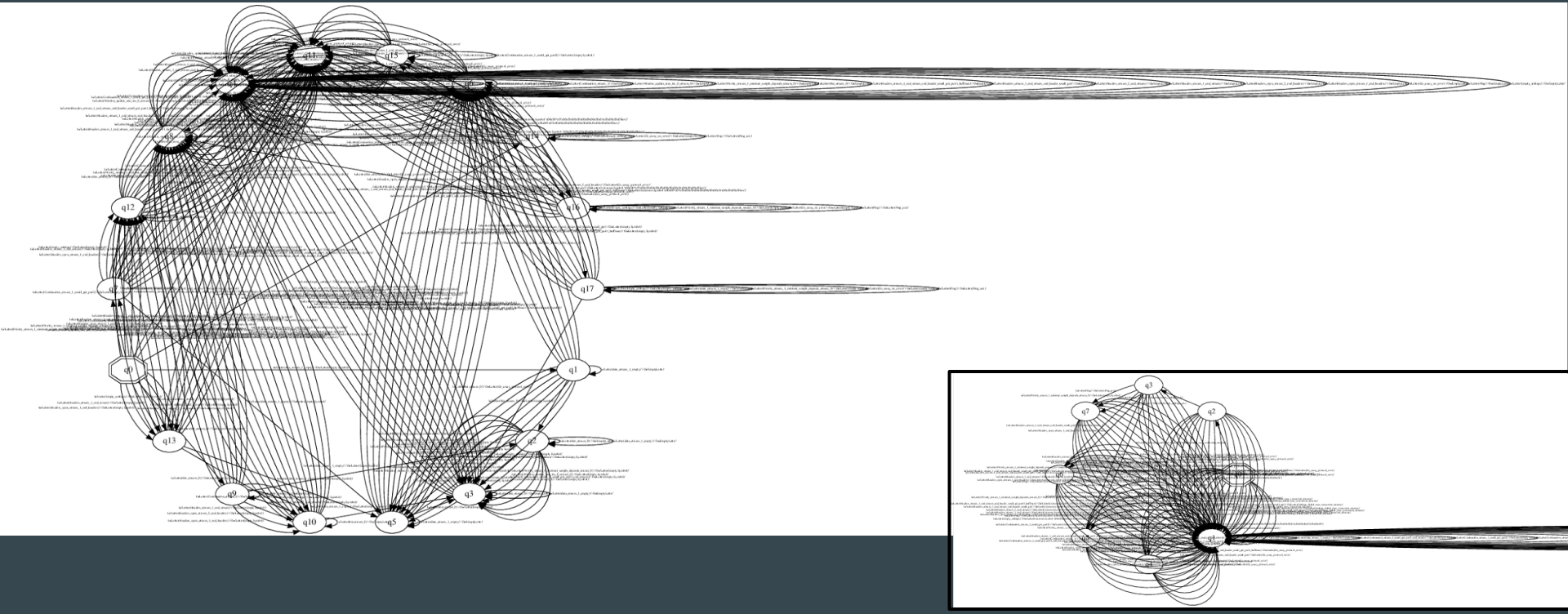
Construction du vocabulaire d'entrée (vocabulaire.py)

- 24 messages d'entrée
- 34 messages de sortie

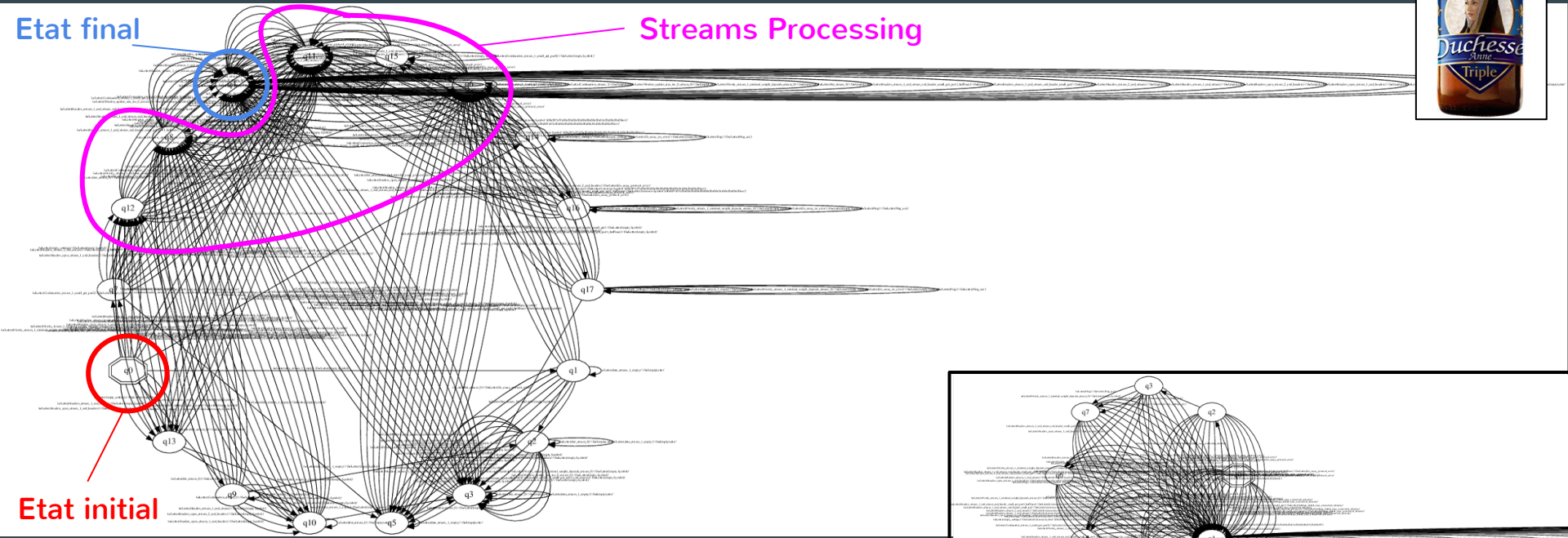
```
600
601 HTTP2_INPUT_VOCABULARY = [
602     CONNECTION_PREFACE,
603     CONNECTION_PREFACE_UNKNOWN_VERSION,
604     EMPTY_SETTINGS,
605     SETTINGS_MAXIMAL_HEADER_TABLE_SIZE,
606     SETTINGS_ENABLE_PUSH,
607     SETTINGS_DISABLE_PUSH,
608     PING,
609     GO_AWAY_NO_ERROR,
610     HEADERS_OPEN_STREAM_1_END_HEADERS,
611     HEADERS_OPEN_STREAM_2_END_HEADERS,
612     HEADERS_STREAM_1_END_STREAM,
613     HEADERS_STREAM_2_END_STREAM,
614     HEADERS_STREAM_1_END_STREAM_END_HEADER_SMALL_GET,
615     HEADERS_STREAM_1_END_STREAM_SMALL_GET_PART1_HUFFMAN,
616     RST_STREAM_0,
617     RST_STREAM_1_LARGE_ERROR_CODE,
618     RST_STREAM_1_FLAGS,
619     PRIORITY_STREAM_1_MINIMAL_WEIGHT_DEPENDS_STREAM_0,
620     WINDOW_UPDATE_SIZE_INC_0_STREAM_0,
621     CONTINUATION_STREAM_1,
622     CONTINUATION_STREAM_1_SMALL_GET_PART2,
623     CONTINUATION_STREAM_1_SMALL_GET_PART3_END_STREAM,
624     DATA_STREAM_1_EMPTY,
625     HEADERS_STREAM_3_END_STREAM_BIG_GET_PART1_HUFFMAN,
626 ]
```



Exemple de résultat



Exemple de résultat



Au programme

1) Présentation rapide du protocole HTTP/2
2) Comparaison de piles serveurs
3) **Exploitation des résultats**

Attaques

Avec la connaissance des automates on peut

- **créer un smart-fuzzer**

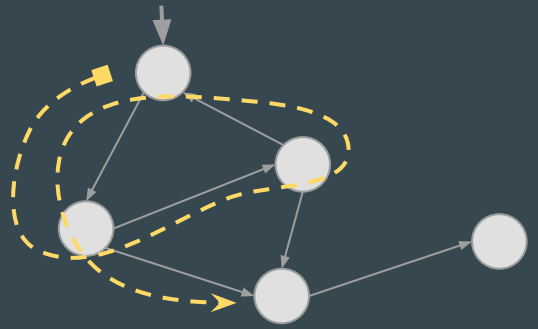
Si les automates ne sont pas strictement équivalents, on peut

- **créer un outil de fingerprint**
- **créer des règles d'évasion d'IDS**

Fuzzer intelligent

Mise en oeuvre d'un smart-fuzzer HTTP/2

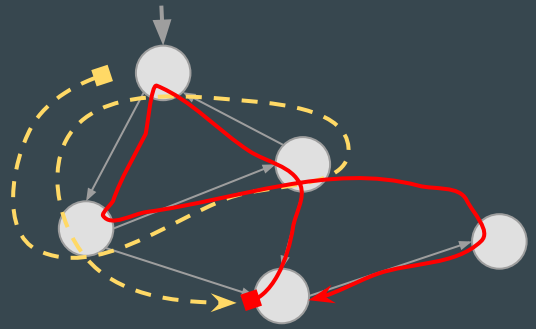
- **Phase 1 : Positionnement dans l'automate**
  - Parcours aléatoire de l'automate
    - Respect du vocabulaire et de la grammaire
    - Profondeur et Reset aléatoire



Fuzzer intelligent

Mise en oeuvre d'un smart-fuzzer HTTP/2

- **Phase 1 : Positionnement dans l'automate**
  - Parcours aléatoire de l'automate
    - Respect du vocabulaire et de la grammaire
    - Profondeur et Reset aléatoire
- **Phase 2 : Fuzzing Grammatical + Vocabulaire**
  - Fuzzing Grammatical
    - Choix aléatoire du prochain état
    - Choix du message parmi les messages acceptés par les transitions menant à cet état
  - Fuzzing Vocabulaire
    - Modification du type et de la valeur des champs
    - Modification des contraintes sur les champs, ...



Fuzzer intelligent

Nécessite du CPU, beaucoup de CPU...

- Ça tourne en tache fond sur mon serveur @home

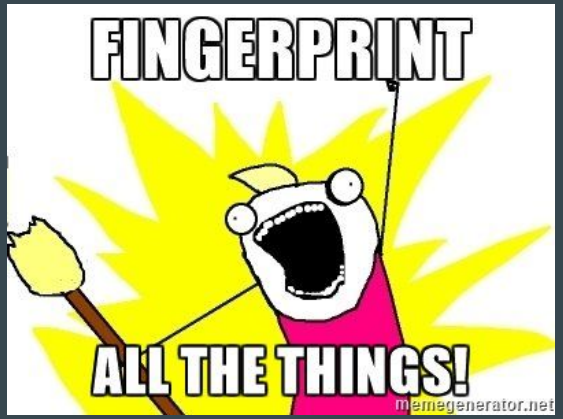
Plusieurs bugs trouvés

- deux bugs de sécurité identifiés sur h2o
- des plantages nginx et apache pas encore caractérisés

TODO

- améliorer l'exploitation des bugs
- utiliser un cloud (AWS, Azure, ...)
- implémenter une boucle de rétro-action

Fingerprint HTTP2



Le fingerprint de serveur web repose

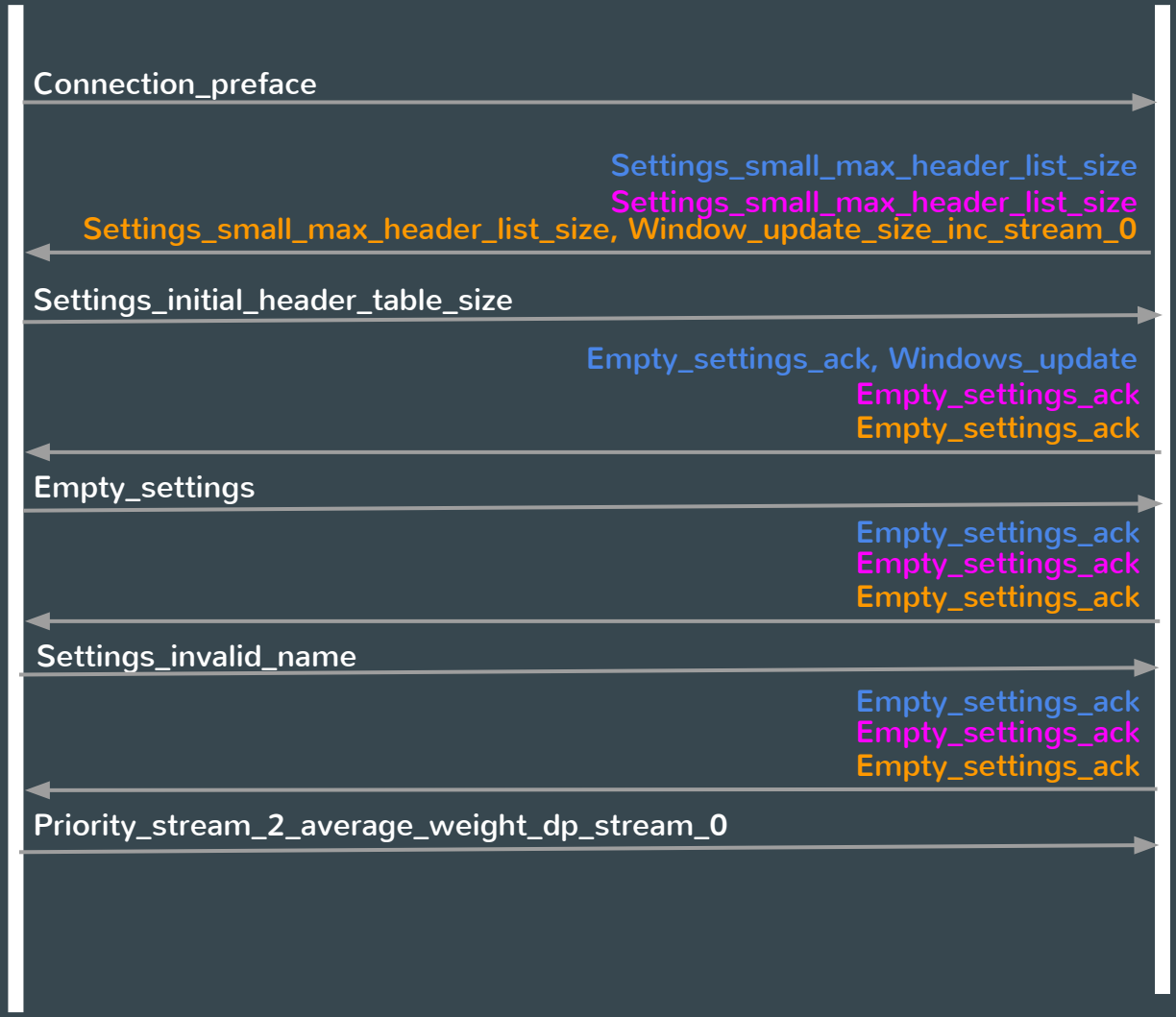
- sur la bannière
- sur la valeur de certains champs

Proposition : utiliser l'automate comme fingerprint

- Identifier des différences entre les automates
- Mise en oeuvre de probabilités

WTF (1) !

- Nginx
- H2o
- Tomcat



Evasion d'IDS

La théorie

- Si des serveurs WEB se comportent différement, difficile pour l'IDS de suivre les flux.

En pratique

- HTTP/2 pas encore proposé par les principaux IDS
- **MAIS, ça risque d'être compliqué pour les dev. d'IDS !**

WTF (2) !

- Nginx
- H2o
- Tomcat

Connection_preface

Settings_small_max_header_list_size
Settings_small_max_header_list_size
Settings_small_max_header_list_size, Window_update_size_inc_stream_0

Settings_large_header_table_size

Empty_settings_ack, Windows_update
Empty_settings_ack
Empty_settings_ack

Priority_stream_1_maximal_weight_depends_stream_1

Empty_symbol
GO-AWAY
Rst_stream_1_protocol_error

Empty_settings_unknown_stream

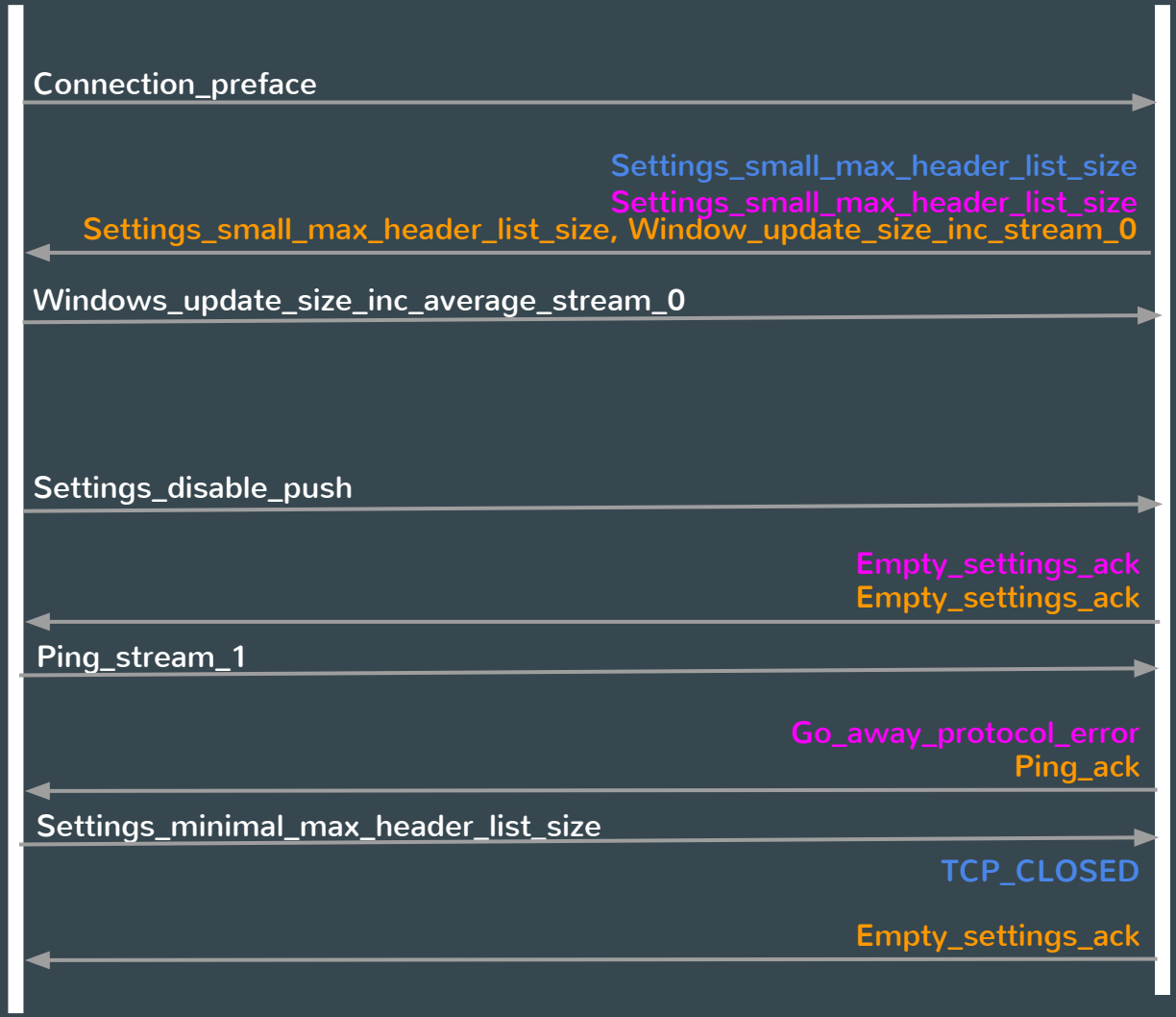
Empty_settings_ack

Ping

TCP_CLOSED
TCP_CLOSED
Ping_ack

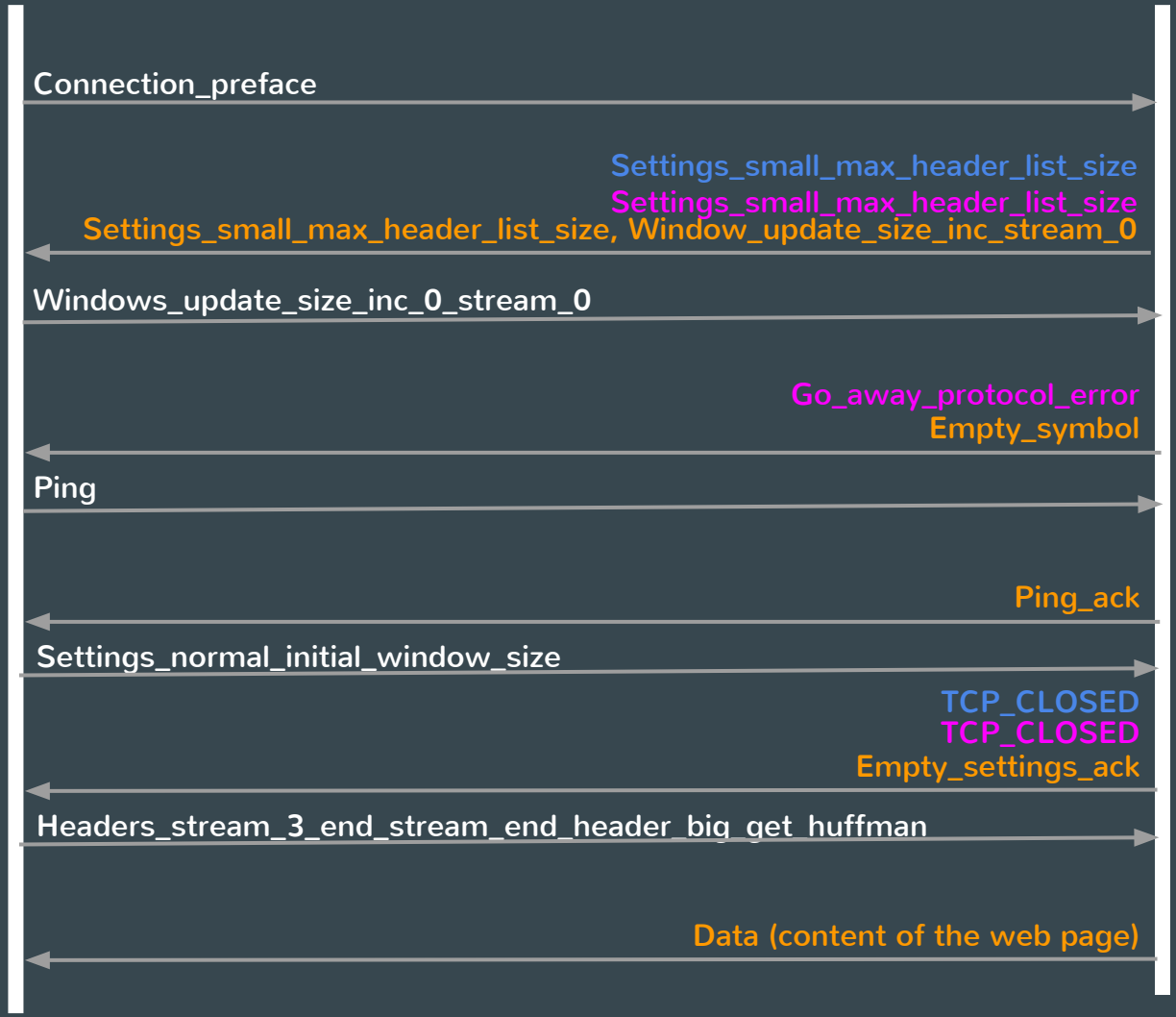
WTF (n) !

- Nginx
- H2o
- Tomcat



WTF (n+1) !

- Nginx
- H2o
- Tomcat



Conclusion

Les RFC ne permettent pas de définir formellement l'automate d'un protocole

- quelques initiatives (cf. Cosmogol par S. Bortzmeyer)

Les outils présentés ici sont disponibles sous license GPLv3:

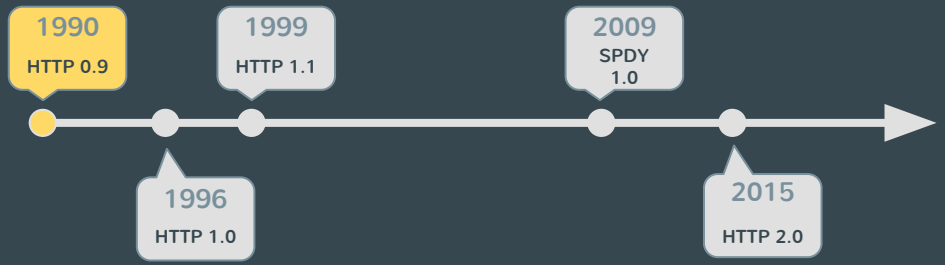
https://github.com/gbossert/pylstar

https://github.com/netzob/netzob

https://github.com/gbossert/http2_compare

N'hésitez pas à venir discuter de vos cas d'application (et de vos questions)

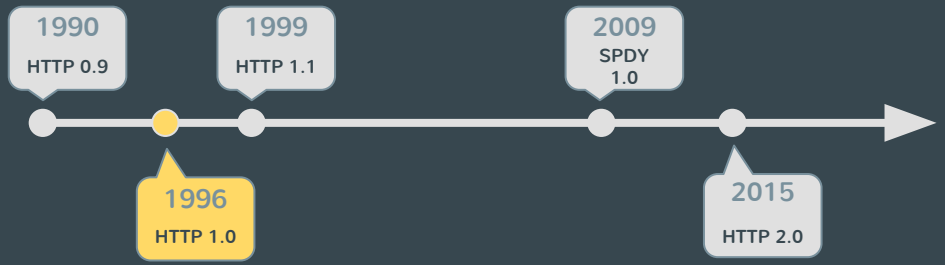
HTTP 0.9



Première version “documentée”

- Une seule méthode disponible : GET
- Pas de notion d’entête ni de metadata
- Un seul type de fichier : text/plain
- RFC 7230 : “The expectation to support HTTP/0.9 requests has been removed”

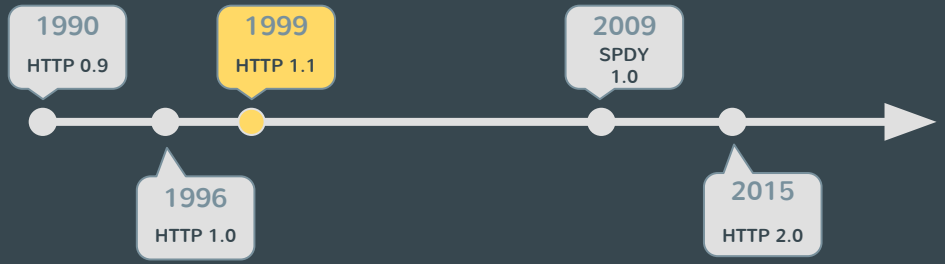
HTTP 1.0



Première version normalisée (RFC 1945)

- Bienvenue au transfert de données (méthode POST)
- Notion de type MIME : possibilité de transporter plusieurs types de fichiers
- Solution pour la gestion du cache et de la congestion
- Authentification : méthodes BASIC et DIGEST
- (...)

HTTP 1.1



Modernisation de HTTP

- Permet l'hébergement virtuel (Virtual-hosting)
- Méthode OPTION + entête **UPGRADE**
- Les connexions TCP sont par défaut persistantes
- Pipelining
- Cache **++** (Cache-Control, age, max-age, ...)
- (...)