HTTP/2

Jeszenszky Péter
Debreceni Egyetem, Informatikai Kar
jeszenszky.peter@inf.unideb.hu

Utolsó módosítás: 2016. április 3.

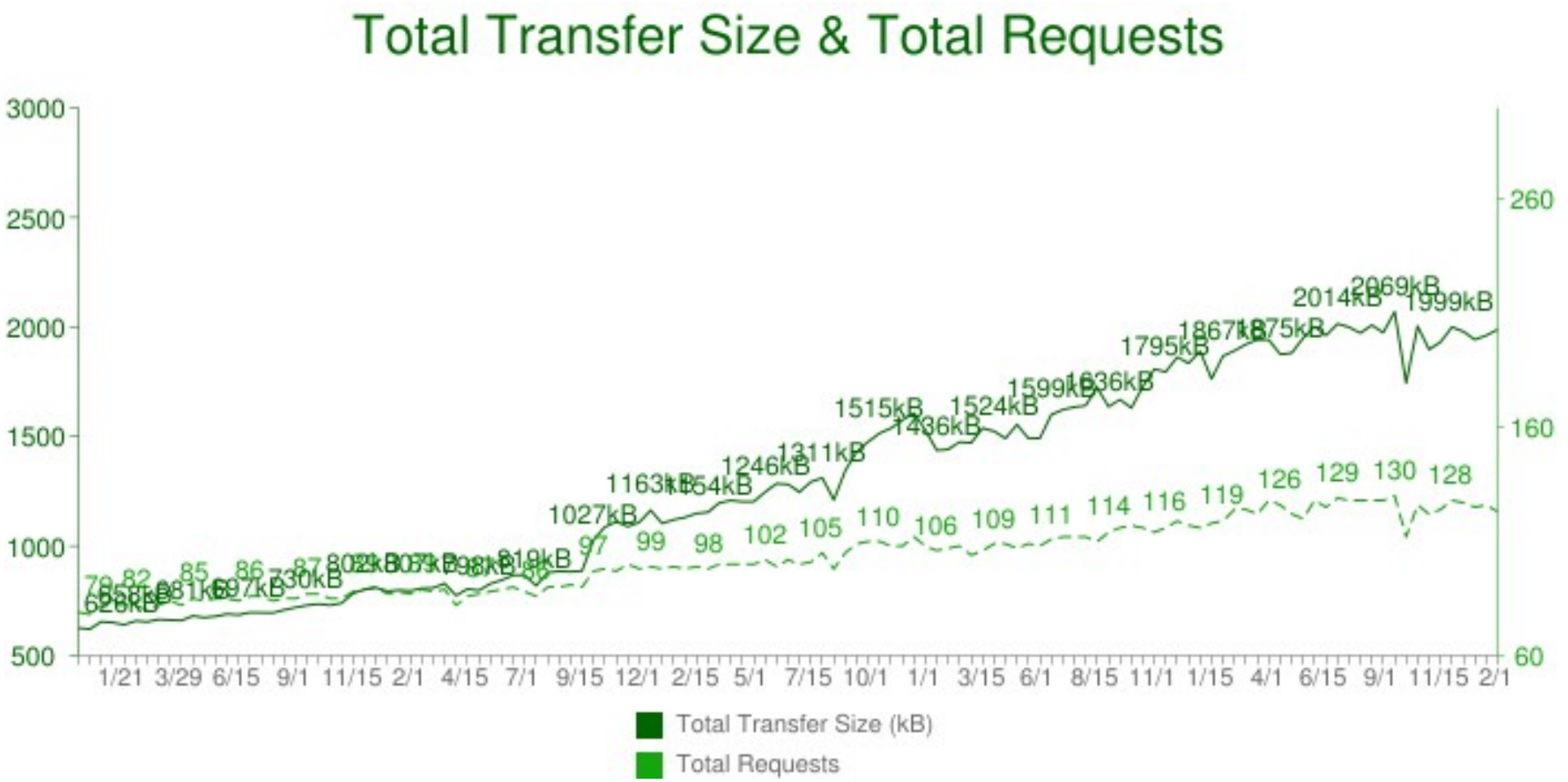
A HTTP/1.*x* a teljesítményt rontó jellemzői (1)

- A HTTP/1.0 egyetlen nem kiszolgált kérést engedett meg egy TCP kapcsolaton

A HTTP/1.*x* a teljesítményt rontó jellemzői (2)

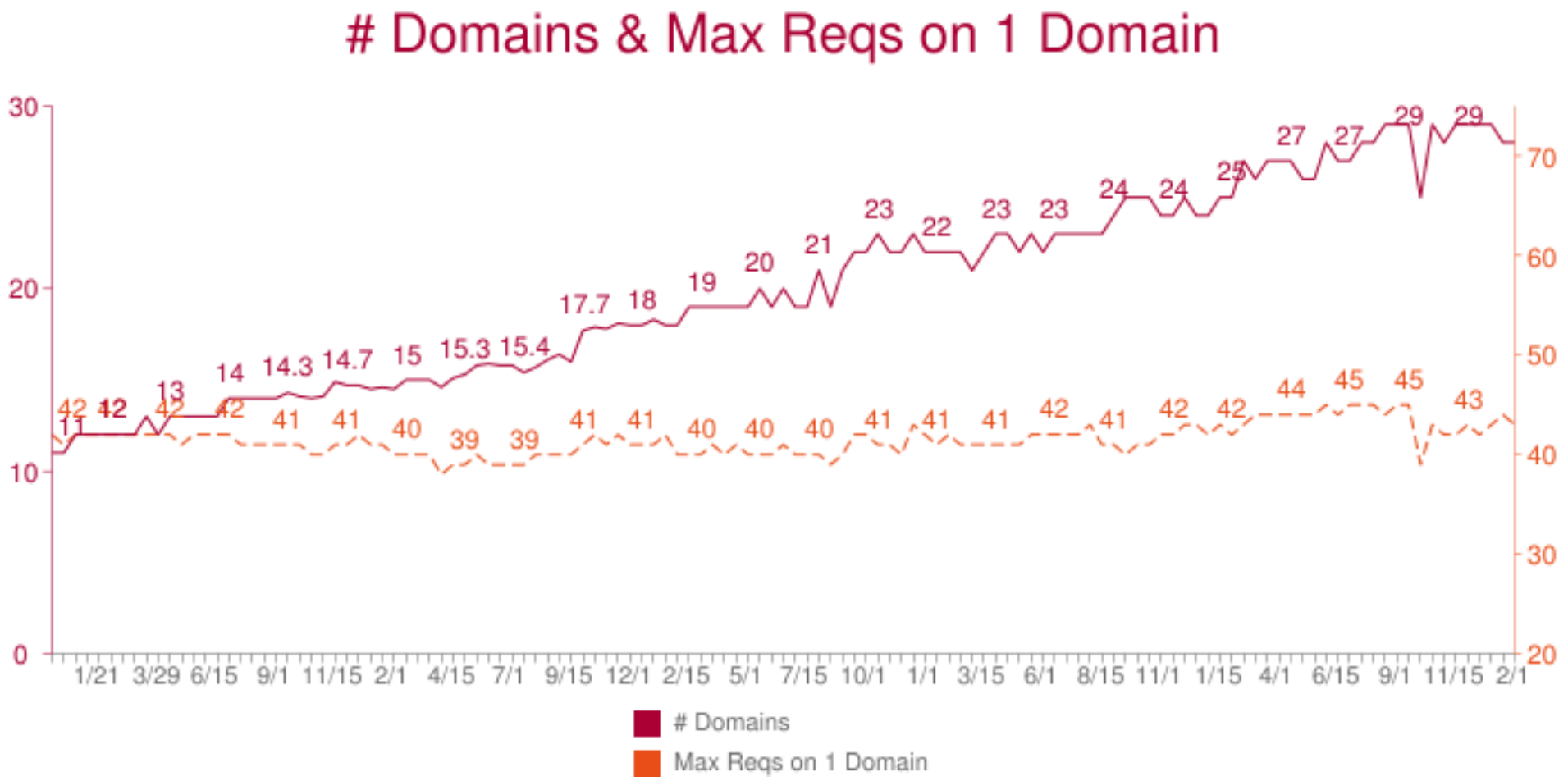
- A HTTP/1.1 bevezette a kérések sorozatban való továbbítását (*pipelining*), de ez csak részben foglalkozott a kérések párhuzamosságával és továbbra is felmerül a sor eleji blokkolás (*head-of-line blocking*) problémája
  - Emiatt azok a HTTP kliensek, melyeknek sok kérést kell végrehajtaniuk, több kapcsolatot használnak egy szerverhez a párhuzamosság eléréséhez és a késleltetési idő csökkentéséhez
- Ráadásul a HTTP fejlécmezők gyakran ismétlődőek és bőbeszédűek, mely szükségtelen hálózati forgalmat okoz és a kezdeti TCP torlódási ablak gyors megtelését eredményezi

Statisztikák (1)



Az átlagos teljes átviteli méret és kérések átlagos száma az 1000 legnépszerűbb webhelynél (2010. november 15. – 2016. február 1.). Forrás: http://httparchive.org/trends.php.

Statisztikák (2)



A 1000 legnépszerűbb webhelyre számított statisztikák (2010. november 15. – 2016. február 1.). Forrás: http://httparchive.org/trends.php.

Megoldások a késleltetési idő csökkentésére (1)

- ***Inline képek***: képek beágyazása CSS stíluslapokba
  - ```
    {
      background: url(
        ⟨Base64-kódolt adatok⟩) no-repeat;
    }
    ```
  - Lásd: Chris Coyier, *Data URIs*, March 25, 2010. https://css-tricks.com/data-uris/
  - Példa: Mozilla Developer Network https://developer.mozilla.org/
    - Lásd a logót a jobb felső sarokban
    ```

Megoldások a késleltetési idő csökkentésére (2)

- **Spriting**: több kép kombinálása egyetlen képállományban
  - Lásd: Chris Coyier, CSS Sprites: *What They Are, Why They're Cool, and How To Use Them*, October 24, 2009. https://css-tricks.com/css-sprites/
  - Példa: Mozilla Developer Network https://developer.mozilla.org/
    - https://developer.mozilla.org/media/redesign/img/logo_sprite.svg

Megoldások a késleltetési idő csökkentésére (3)

- **Sharding**: a tartalom elosztása több szerveren
  - Érdemes lehet például a képeket egy olyan webszerveren elhelyezni, mely egyáltalán nem használ sütiket
- **Összefűzés**: több CSS stíluslap és JavaScript állomány összefűzése
- **Minification**: felesleges karakterek eltávolítása CSS stíluslapokból és JavaScript állományokból anélkül, hogy az erőforrás a böngésző általi feldolgozása módosulna
  - Például megjegyzések és *whitespace* karakterek eltávolítása

HTTP/2

- A HTTP/2 a HTTP szemantika egy hatékonyabb kifejezési módja
  - Cél a hálózati erőforrások hatékonyabb használata és a végfelhasználó által megfigyelhető késleltetési idő csökkentése
  - Az egyik fő cél, hogy lehetővé tegye a böngészők számára egy webhelyhez egyetlen kapcsolat használatát
- Ugyanazokat a metódusokat, állapotkódokat, fejlécmezőket és URI sémákat használja, mint a HTTP/1.1
  - Az üzenetek formálása és átvitele történik eltérő módon

Fejlesztés

- Az IETF HTTP munkacsoportja (HTTPbis) fejleszti
- Honlap: http://http2.github.io/

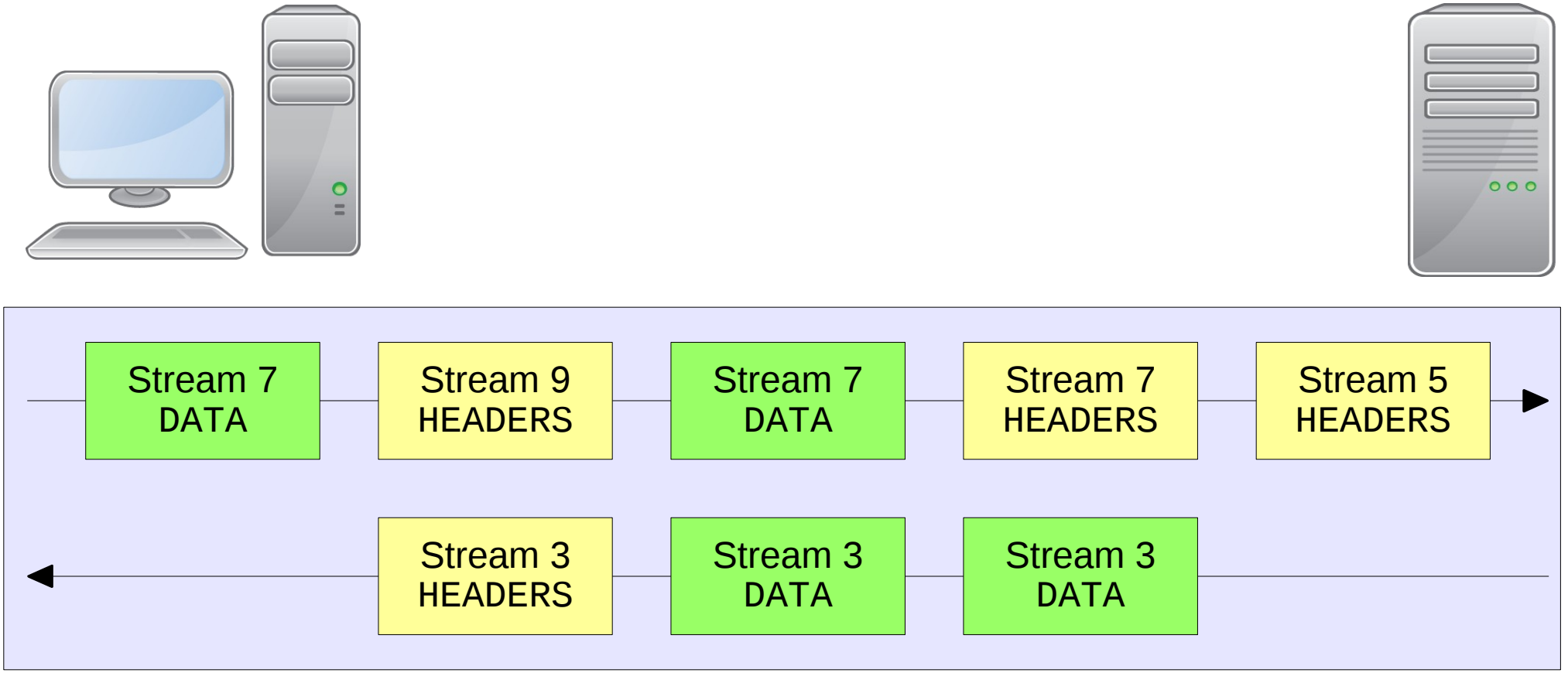
Specifikációk

- Mike Belshe, Roberto Peon, Martin Thomson (ed.), *Hypertext Transfer Protocol Version 2 (HTTP/2)*, RFC 7540, May 2015. https://tools.ietf.org/html/rfc7540
- Roberto Peon, Herve Ruellan, *HPACK: Header Compression for HTTP/2*, RFC 7541, May 2015. https://tools.ietf.org/html/rfc7541

A HTTP/2 újdonságai

- **Multiplexelés**: megvalósítás adatfolyamok használatával
  - Az adatfolyamok túlnyomórészt egymástól függetlenek, így egy blokkolt vagy beragadt adatfolyam nem akadályozza az előrehaladást más adatfolyamokkal
- ***Forgalomvezérlés*** és **rangsorolás**: a multiplexelt adatfolyamok hatékony használatát biztosító mechanizmusok
  - A forgalomvezérlés biztosítja, hogy a fogadónak annyi adat kerül továbbításra, amennyit az kezelni tud
  - A rangsorolás biztosítja, hogy a korlátozottan rendelkezésre álló erőforrások először a legfontosabb adatfolyamokhoz rendelhetők hozzá
- **Szerver *push***: lehetővé teszi a szervernek, hogy spekulatív módon küldjön olyan adatokat egy kliensnek, melyekre annak előreláthatólag szüksége lesz
- **Bináris protokoll**: az üzenetek hatékonyabb feldolgozását teszi lehetővé az üzenetek bináris formálása
- **Fejlécmezők tömörítése** (**HPACK**)

Üzenet multiplexelés



Előzmény

- A HTTP/2 a Google által fejlesztett SPDY protokollon alapul http://www.chromium.org/spdy
  - A célkitűzéseket lásd: *SPDY: An experimental protocol for a faster web* http://www.chromium.org/spdy/spdy-whitepaper
    - A célkitűzések között szerepelt például az oldalak betöltési idejének felére csökkentése
  - Az SPDY/2 szolgált alapul a HTTP/2 specifikációhoz
    - Verziótörténet: https://sites.google.com/a/chromium.org/dev/spdy/spdy-protocol

SPDY implementációk

- Kliensek:
  - **Firefox** (**Gecko**): lásd a `network.http.spdy.enabled` opciót (`about:config`)
  - **Chromium** (**Blink**), **Opera** (**Blink**):
    - Lásd: `chrome://net-internals/#spdy`
  - **Internet Explorer** (**Trident**)
    - Lásd: *What's New in Internet Explorer 11* https://technet.microsoft.com/en-us/ie/dn269977.aspx
  - Microsoft Edge (EdgeHTML):
    - Eltávolították a SPDY támogatást, lásd: *HTTP/2 replaces SPDY/3* https://msdn.microsoft.com/en-us/library/dn905221%28v=vs.85%29.aspx
- Szerverek:
  - **Apache HTTP Server** https://code.google.com/p/mod-spdy/ https://svn.apache.org/repos/asf/httpd/mod_spdy/
  - **nginx** http://nginx.org/en/docs/http/ngx_http_spdy_module.html

SPDY-képes webhelyek

- Például:
  - Dropbox https://www.dropbox.com/
  - Facebook https://www.facebook.com/
  - Google
  - reddit https://www.reddit.com/
  - Twitter https://twitter.com/
  - Wikipedia https://www.wikipedia.org/
  - Yahoo! https://www.yahoo.com/
- SPDY támogatás ellenőrzése:
  - SPDYCheck.org https://spdycheck.org/

HTTP/2 implementációk (1)

- Az implementációk egy felsorolását lásd itt: https://github.com/http2/http2-spec/wiki/Implementations

HTTP/2 implementációk (2)

- Böngészők: a felsorolt böngészők mindegyikében csak TLS-en keresztüli HTTP/2 támogatás (`https` URI-k)
  - **Firefox** (**Gecko**): a 34-es verzió óta
    - Lásd: *Networking/http2* https://wiki.mozilla.org/Networking/http2
    - Lásd a `network.http.spdy.enabled.http2` és `network.http.spdy.enabled.http2draft` opciókat (`about:config`)
  - **Chromium** (**Blink**):
    - Lásd az `#enable-spdy4` opciót (`chrome://flags`) http://www.chromium.org/spdy/http2
    - Lásd még: *Hello HTTP/2, Goodbye SPDY*, February 9, 2015. http://blog.chromium.org/2015/02/hello-http2-goodbye-spdy-http-is_9.html
  - **Opera** (**Blink**):
    - Lásd az `#enable-spdy4` opciót (`opera://flags`)
  - **Internet Explorer** (**Trident**):
    - Csak ebben a kiadásban: Internet Explorer 11, Windows 10 Technical Preview
  - **Microsoft Edge (EdgeHTML)**:
    - Lásd: *HTTP/2 replaces SPDY/3* https://msdn.microsoft.com/en-us/library/dn905221%28v=vs.85%29.aspx
- Lásd: Martin Brinkmann, *The state of HTTP/2 in Firefox, Chrome and Internet Explorer*, February 19, 2015. http://www.ghacks.net/2015/02/19/the-state-of-http2-in-firefox-chrome-and-internet-explorer/

HTTP/2 implementációk (3)

- Egyéb kliensek:
  - cURL (programozási nyelv: C, licenc: X11 License) http://curl.haxx.se/
    - HTTP/2 támogatáshoz forrásból lehet szükséges fordítani és telepíteni
  - Hyper (programozási nyelv: Python, licenc: MIT License) https://github.com/lukasa/hyper
  - Netty (programozási nyelv: Java, licenc: Apache License 2) http://netty.io/
  - nghttp2 (programozási nyelv: C, licenc: MIT License) https://nghttp2.org/

HTTP/2 implementációk (4)

- Szerverek:
  - Apache HTTP Server (programozási nyelv: C, licenc: Apache License 2) https://httpd.apache.org/
    - A 2.4.18 verzióban megjelent `mod_http2` modul biztosít HTTP/2 támogatást: *Overview of new features in Apache HTTP Server 2.4* http://httpd.apache.org/docs/trunk/new_features_2_4.html
  - Jetty (programozási nyelv: Java, licenc: Apache License 2/Eclipse Public License 1.0) http://eclipse.org/jetty/
    - HTTP/2 támogatás a központi Maven tárolóból elérhető `9.3.0.M2` verzióban
  - nghttp2 (programozási nyelv: C, licenc: MIT License) https://nghttp2.org/
  - nginx:
    - *Faisal Memon, NGINX Open Source 1.9.5 Released with HTTP/2 Support*, September 22, 2015. https://www.nginx.com/blog/nginx-1-9-5/
  - OpenLiteSpeed (programozási nyelv: C++, licenc: GNU GPL v3) http://open.litespeedtech.com/
  - Undertow (programozási nyelv: Java, licenc: GNU GPL v2.1) https://http2.undertow.io/
    - A WildFly (korábbi nevén JBoss) alkalmazásszerver webszervere http://wildfly.org/

Hivatalos Java támogatás

- Kliens oldal:
  - A JDK 9-ben várható egy új HTTP kliens API, mely implementálja a HTTP/2-t és helyettesítheti a `java.net.HttpURLConnection` osztályt
  - Lásd: *JEP 110: HTTP 2 Client* http://openjdk.java.net/jeps/110
- Szerver oldal:
  - A Servlet 4.0 API fogja támogatni, mely a Java EE 8-ban várható 2016 harmadik negyedévében
  - Lásd: David Delabassee, *Servlet 4.0*, July 24, 2014. https://blogs.oracle.com/theaquarium/entry/servlet_4_0
- Lásd még:
  - Edward Burns, HTTP/2 comes to Java – What Servlet 4.0 means to you, DevNexus 2015, March 10–12, 2015. http://www.slideshare.net/edburns/http2-comes-to-java-what-servlet-40-means-to-you-devnexus-2015

Böngésző kiegészítők

- Böngésző kiegészítők, melyek jelzik, ha a szerverrel való kommunikáció az SPDY vagy HTTP/2 protokoll szerint történik:
  - **Firefox**: HTTP/2 and SPDY indicator https://addons.mozilla.org/hu/firefox/addon/spdy-indicator/
  - **Chromium**: Chrome SPDY indicator http://www.devthought.com/2012/03/10/chrome-spdy-indicator/
  - **Opera**: SPDY indicator https://addons.opera.com/hu/extensions/details/spdy-indicator/

HTTP/2-képes webhelyek

- Facebook https://www.facebook.com/
- Flickr https://www.flickr.com/
- Google
- Twitter https://twitter.com/
- Yahoo https://www.yahoo.com/

nghttp2

- C-ben írt HTTP/2 implementáció, az egyik legérettebb
  - C programkönyvtár (`libnghttp2`)
  - Kliens (`nghttp`)
  - Szerver (`nghttpd`)
  - Fordított proxy (`nghttpx`)
- Példa a kliens használatára:
  - `nghttp -v --stat https://nghttp2.org/`

cURL

- HTTP/2 támogatáshoz forrásból lehet szükséges fordítani és telepíteni
  - Lásd:
    - *HTTP/2 with curl* http://curl.haxx.se/dev/readme-http2.html
    - *Curl With HTTP2 Support* https://serversforhackers.com/video/curl-with-http2-support
- Példa a program használatára:
  - `curl --http2 -v http://nghttp2.org/`

Fogalmak

- **Kliens**: egy HTTP/2 kapcsolatot létesítő végpont, a kliensek HTTP kéréseket küldenek és HTTP válaszokat kapnak
- **Szerver**: egy HTTP/2 kapcsolatot elfogadó végpont, a szerverek HTTP kéréseket fogadnak és HTTP válaszokat küldenek
- **Végpont** (***endpoint***): a klienst vagy a szervert jelenti
- **Kapcsolat**: két végpont közötti átviteli rétegbeli kapcsolat
- **Keret** (***frame***): a legkisebb kommunikációs egység egy HTTP/2 kapcsolaton belül
- **Adatfolyam** (***stream***): egy HTTP/2 kapcsolaton belül a kliens és a szerver között váltott keretek egy független, kétirányú sorozata

HTTP/2 kapcsolat létrehozása (1)

- A TCP kapcsolatot a kliens kezdeményezi
- A kliensnek először ki kell derítenie, hogy a szerver támogatja-e a HTTP/2-t, ez eltérően történik `http` és `https` URI-knál
  - Egy `http` URI esetén a kliens az `Upgrade` fejlécmezőt használja egy HTTP/1.1 kérésben
  - Egy `https` URI esetén az ALPN révén történik a protokoll egyeztetése

HTTP/2 kapcsolat létrehozása (2)

- Példa:
  - `curl --http2 -v http://nghttp2.org/`

```
> GET / HTTP/1.1
> Host: nghttp2.org
> User-Agent: curl/7.47.1
> Accept: */*
> Connection: Upgrade, HTTP2-Settings
> Upgrade: h2c
> HTTP2-Settings: AAMAAABkAAQAAP__
>
< HTTP/1.1 101 Switching Protocols
< Connection: Upgrade
< Upgrade: h2c
<
< HTTP/2.0 200
...
```

HTTP/2 kapcsolat létrehozása (3)

- Mindkét végpontnak egy **kapcsolat bevezetőt** (***connection preface***) kell küldenie a használt protokoll végső megerősítéseként és a kapcsolat kezdeti beállításainak megállapításához
  - A kliens bevezetője a `"PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n"` oktettsorozattal kezdődik, melyet egy `SETTINGS` keret követ, mely lehet üres
    - A bevezető választásáról lásd: Matthew Kerwin, *Painting Sheds*, 2013-12-09. http://matthew.kerwin.net.au/blog/20131209_painting_sheds
  - A szerver bevezetője egy potenciálisan üres `SETTINGS` keretből áll
- A bevezetőt a kliens közvetlenül a `101` (`Switching Protocols`) válasz fogadása után küldi el vagy a TLS kapcsolat első alkalmazási adatoktettjeiként
- Ha a kliens tudja, hogy a szerver támogatja a protokollt, akkor a kapcsolat létrehozásakor küldi el a bevezetőt

ALPN (1)

- A Transport Layer Security (TLS) egy kiterjesztése alkalmazási-réteg protokoll egyeztetéséhez:
  - Stephan Friedl, Andrei Popov, Adam Langley, Emile Stephan, *Transport Layer Security (TLS) – Application-Layer Protocol Negotiation Extension*, RFC 7301, July 2014. http://tools.ietf.org/html/rfc7301
- A protokoll azonosítók regisztrálását az IANA végzi
  - Lásd: *Application-Layer Protocol Negotiation (ALPN) Protocol Ids* http://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml#alpn-protocol-ids

ALPN (2)

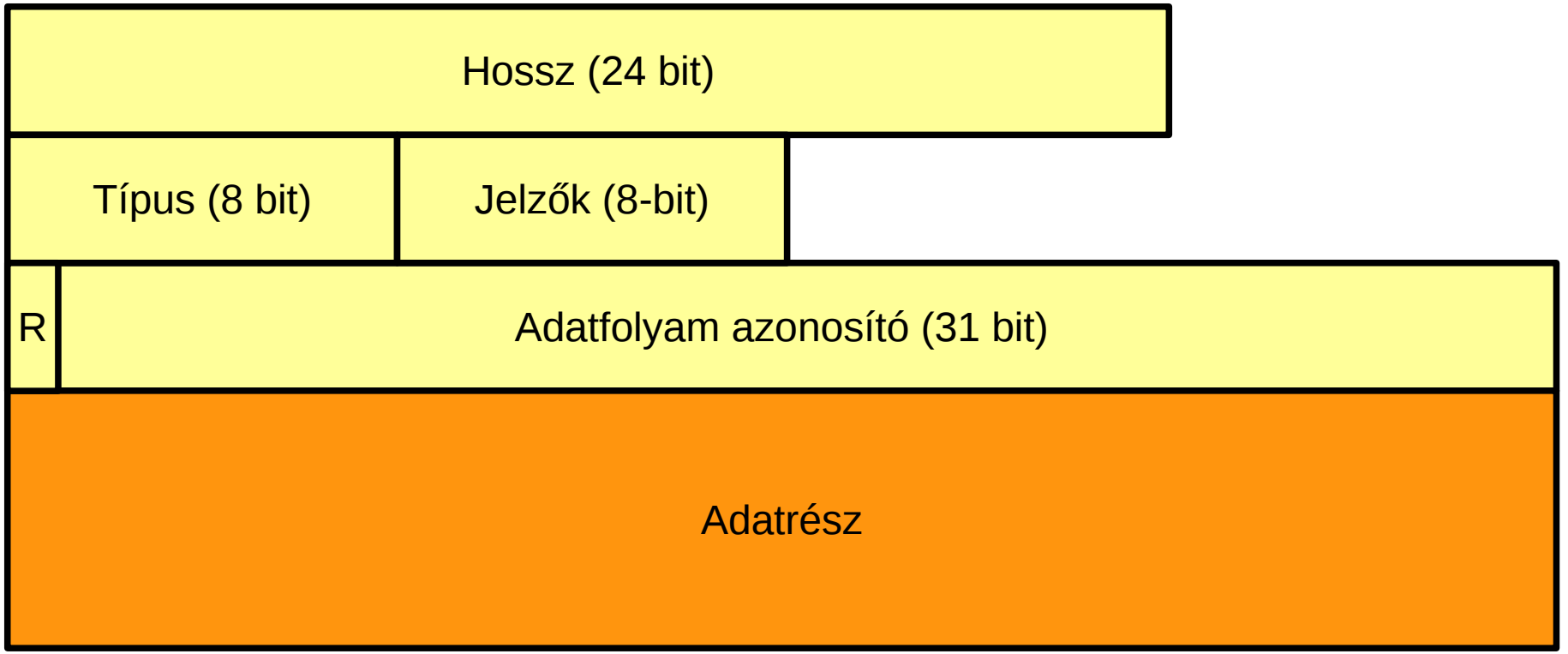
- Protokoll egyeztetés:
  - A kliens a TLS `ClientHello` üzenet részeként elküldi a szervernek az általa támogatott protokollok listáját
  - A szerver kiválaszt egy protokollt, melyet a TLS `ServerHello` üzenet részeként küld vissza a kliensnek
- Az alkalmazási protokoll egyeztetése így a TLS kézfogás során történik

ALPN (3)

- A HTTP/2 által használt protokoll azonosítók:
  - `"h2"`: HTTP/2 TLS-en keresztül
  - `"h2c"`: HTTP/2 TCP-n keresztül (ahol `'c'` a *cleartext* kifejezést jelenti)

Keretek felépítése (1)

- Minden keret egy 9 oktett méretű fejléccel kezdődik, melyet egy változó hosszú **adatrész** (***payload***) követ



Keretek felépítése (2)

- A keret fejléc mezői:
  - **Hossz**: az adatrész hosszát szolgáltató 24-bites előjel nélküli egész
  - **Típus**: a keret 8 biten ábrázolt típusa, mely meghatározza a keret felépítését és szemantikáját
  - **Jelzők**: 8 biten ábrázolt logikai jelzők, melyek jelentése a keret típusától függ
  - **R**: fenntartott célú 0 értékű bit, melynek nincs definiált jelentése
  - **Adatfolyam azonosító**: egy adatfolyamot azonosító 31-bites előjel nélküli egész
- Az adatrész felépítése és tartalma a keret típusától függ

Keretek mérete

- A fogadó `SETTINGS_MAX_FRAME_SIZE` beállításának értéke határozza meg a keretek adatrészének maximális méretét
  - Értéke 2^{14} és $2^{24}–1$ között kell, hogy legyen
- Bizonyos fajta keretek (például a `PING`) további korlátokat szabnak a keretek adatrészének méretére

Keretek fajtái (1)

Kód	Típus	Funkció
0x0	DATA	Üzenet *payload* átvitele
0x1	HEADERS	Adatfolyam megnyitása és egy fejléc blokk töredék továbbítása
0x2	PRIORITY	Adatfolyam prioritásának előírása
0x3	RST_STREAM	Adatfolyam azonnali megszüntetése
0x4	SETTINGS	Konfigurációs paraméterek továbbítása, vétel nyugtázása

Keretek fajtái (2)

Kód	Típus	Funkció
0x5	PUSH_PROMISE	Szerver *push* megvalósítása
0x6	PING	Minimális körbefordulási idő (RRT) mérése, tétlen (*idle*) állapotú adatfolyam működőképességének megállapítása
0x7	GOAWAY	Azt jelzi a másik fél felé, hogy hagyja abba adatfolyamok létrehozását a kapcsolaton
0x8	WINDOW_UPDATE	Forgalomvezérlés megvalósítása
0x9	CONTINUATION	Fejléc blokk töredékek sorozatának folytatása

Adatfolyamok jellemzői

- Egyetlen HTTP/2 kapcsolat több egyidejűleg nyitott adatfolyamot tartalmazhat
- Az adatfolyamokat egyoldalúan (a másik féllel való egyeztetés nélkül) hozhatja létre a kliens vagy a szerver, de megosztva használhatják őket
- Az adatfolyamokat mindkét végpont lezárhatja
- Lényeges, hogy milyen sorrendben kerülnek elküldésre a keretek egy adatfolyamon
  - A fogadó abban a sorrendben dolgozza fel a kereteket, melyben megkapja őket
- Az adatfolyamokat egy előjel nélküli egész szám azonosítja

Adatfolyamok azonosítása (1)

- Az adatfolyam azonosító egy 31-bites előjel nélküli egész, melyet az adatfolyamot létrehozó végpont rendel az adatfolyamhoz
  - A kliens által megnyitott adatfolyamokat páratlan számok azonosítják
  - A szerver által megnyitott adatfolyamokat páros számok azonosítják
  - A `0x00` adatfolyam azonosítót a kapcsolatot vezérlő üzenetekhez használják
- Egy újonnan létrehozott adatfolyam azonosítója nagyobb kell, hogy legyen a kezdeményező végpont által létrehozott vagy fenntartott adatfolyam azonosítók mindegyikénél

Adatfolyamok azonosítása (2)

- Az adatfolyam azonosítók nem újrafelhasználhatóak
- Hosszú élettartamú kapcsolatoknál kimerülhet a rendelkezésre álló adatfolyam azonosítók tartománya
  - Ilyenkor egy új kapcsolatot kell létrehozni

Forgalomvezérlés (1)

- Az adatfolyamok a TCP kapcsolat használatáért folyó versengést vezetnek be, blokkolt adatfolyamokat eredményezve
- A **forgalomvezérlés** (***flow control***) biztosítja, hogy az ugyanahhoz a kapcsolathoz tartozó adatfolyamok ne akadályozzák egymást
- Az egyes adatfolyamokra és a kapcsolat egészére is vonatkozik
- Csak a DATA keretek vannak alávetve forgalomvezérlésnek
  - Ez biztosítja azt, hogy a vezérlő kereteket nem blokkolja a forgalomvezérlés

Forgalomvezérlés (2)

- Az olyan végpontokat hivatott védeni, melyek számára korlátozottan állnak rendelkezésre erőforrások (például memória)
- Olyan esetekkel foglalkozik, amikor a fogadó nem képes egy adatfolyamon adatokat feldolgozni, de a kapcsolaton más adatfolyamokon akarja folytatni a feldolgozást

Forgalomvezérlés (3)

- A **forgalomvezérlő ablak** (***flow control window***) egy egész érték, mely azt jelzi, hogy a küldő számára hány oktettnyi adat átvitele engedélyezett
- Minden egyes adatfolyamhoz és a kapcsolathoz is egy külön forgalomvezérlő ablak tartozik

Forgalomvezérlés (4)

- A forgalomvezérlést a fogadó szabályozza, minden egyes adatfolyamhoz és az egész kapcsolathoz tetszőleges ablakméretet állíthat be, melyet egy küldőnek tiszteletben kell tartania
- A kliensek, szerverek és közvetítők egymástól függetlenül teszik közzé, hogy mekkora az ablakméretük fogadóként

Forgalomvezérlés (5)

- Egy HTTP/2 kapcsolat létrehozása után az ablakméret alapértelmezett kezdeti értéke minden új adatfolyamhoz és a teljes kapcsolathoz is $2^{16}–1 = 65\,535$ oktett
  - Az adatfolyamok ablakmérete a `SETTINGS_INITIAL_WINDOW_SIZE` beállítással szabályozható
  - A teljes kapcsolat ablakmérete `WINDOW_UPDATE` keretekkel módosítható
- A legnagyobb lehetséges ablakméret $2^{31}–1$ oktett

Forgalomvezérlés (6)

- Tilos olyan forgalomvezérlésnek alávetett keretet küldése, melynek hossza meghaladja a fogadó az adott adatfolyamhoz vagy a kapcsolathoz tartozó ablakában rendelkezésre álló helyet
- Egy forgalomvezérlésnek alávetett keret elküldése után a küldő mindkét ablakban a keret hosszával csökkenti a rendelkezésre álló helyet
- A keret fogadója egy `WINDOW_UPDATE` keretet küld, amint feldolgozza az adatokat és helyet szabadít fel az ablakokban
  - Külön `WINDOW_UPDATE` keretek kerülnek elküldésre az adatfolyam és a kapcsolat szintű ablakokhoz
- Egy `WINDOW_UPDATE` keretet fogadó küldő a keretben meghatározott mértékben növeli a megfelelő ablakot

Forgalomvezérlés (7)

- A számításoknál nem kerül figyelembe vételre a keretek 9 oktett méretű fejléce
- A HTTP/2 csak a `WINDOW_UPDATE` keret felépítését és jelentését definiálja, nem ad algoritmust a forgalomvezérlés megvalósításához
  - Így például nem határozza meg, hogy egy küldő hogyan dönti el azt, hogy mikor kell `WINDOW_UPDATE` keretet küldenie, vagy hogy mekkora értéket használjon

Adatfolyam prioritás (1)

- Egy kliens egy új adatfolyamhoz a megnyitáskor hozzárendelhet egy prioritást, mely egy 1 és 256 közötti egész
  - A prioritás a `HEADERS` keretben adható meg, később a `PRIORITY` kerettel módosítható
- A prioritás lehetővé teszi egy végpont számára annak jelzését, hogy hogyan szeretné, hogy a másik fél erőforrásokat foglaljon le számára, ha egyidejűleg több adatfolyamot is kezel

Adatfolyam prioritás (2)

- A prioritás felhasználható az adatfolyamok keretek küldéséhez való kiválasztásához, amikor a küldéshez korlátozott kapacitás áll rendelkezésre
- A prioritás csupán egy ajánlás, nem garantál semmiféle feldolgozási vagy átviteli sorrendet más adatfolyamokhoz képest
- A HTTP/2 nem határoz meg algoritmust a prioritások kezeléséhez, csupán lehetőséget ad azok adatfolyamokhoz való rendeléséhez

Adatfolyam függőségek (1)

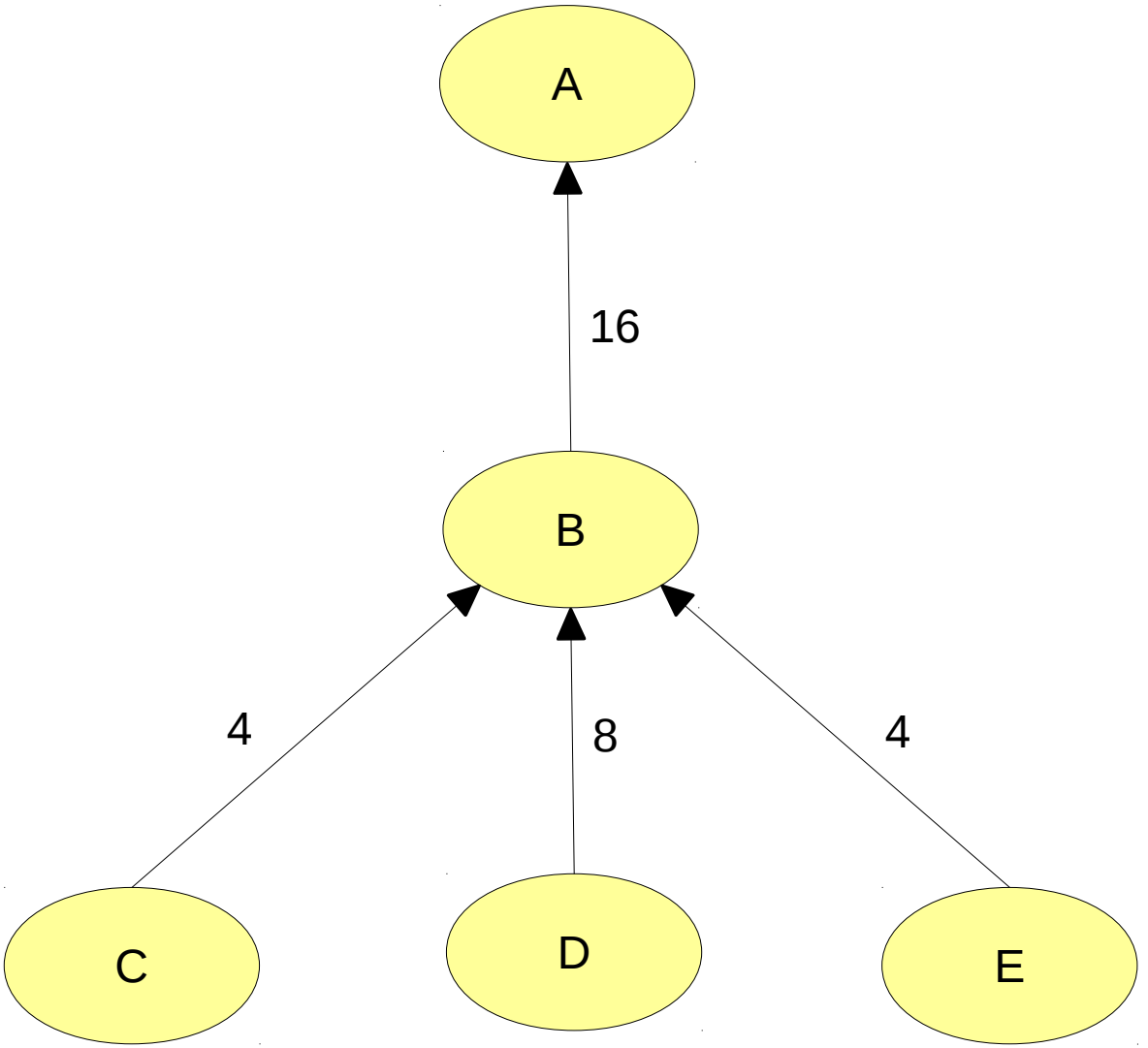
- Minden egyes adatfolyamhoz megadható egy másik adatfolyamtól való függés
  - Minden függőséghez egy relatív súly tartozik
- Egy függőség megadása azt a preferenciát fejezi ki, hogy az erőforrásokat inkább a jelzett adatfolyamhoz kell lefoglalni, nem pedig a függő adatfolyamhoz
- A szülő adatfolyam egy olyan adatfolyam, melytől egy másik adatfolyam függ
  - Más adatfolyamoktól nem függő adatfolyamokhoz a `0x00` szülőt kell megadni

Adatfolyam függőségek (2)

- Egy függőségi fában egy függő adatfolyamhoz csak akkor ajánlott erőforrások lefoglalása, ha vagy lezárásra került az összes olyan adatfolyam, melyektől függ, vagy nem lehetséges haladást elérni rajtuk
- Az ugyanahhoz a szülőhöz tartozó adatfolyamokhoz a súlyokkal arányosan kell az erőforrásokat lefoglalni

Adatfolyam függőségek (3)

- Példa:



Adatfolyam függőségek (4)

- Példa adatfolyam függőségek gyakorlati használatára:
  - *How Dependency Based Prioritization Works* https://nghttp2.org/blog/2014/04/27/how-dependency-based-prioritization-works/

HTTP kérés/válasz váltás

- Egy kliens minden egyes kérést egy új adatfolyamon küld el a szervernek, melyhez egy korábban nem használt adatfolyam azonosítót használ
- A szerver a választ ugyanezen az adatfolyamon küldi vissza
- A kérés/válasz váltás során az adatfolyam teljesen „elhasználódik”
- A választ záró keret lezárja az adatfolyamot

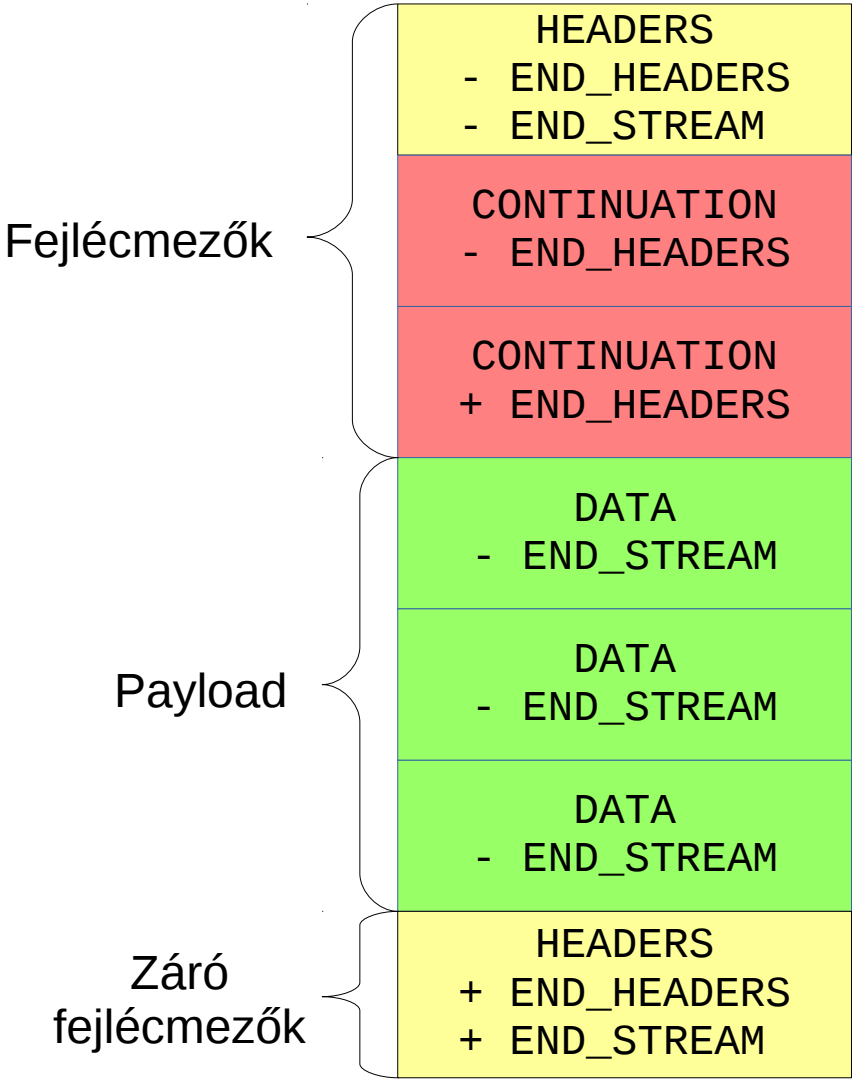
HTTP üzenetek felépítése (1)

- Egy HTTP üzenet a következőkből áll:
  - Csak válaszoknál 0 vagy több `HEADERS` keretből, melyek mindegyikét 0 vagy több `CONTINUATION` keret követi, és melyek 1*xx* állapotkódú válaszok üzenet fejléceit tartalmazzák
  - Egy `HEADERS` keretből és azt követő 0 vagy több `CONTINUATION` keretből, melyek az üzenet fejléceket tartalmazzák
  - 0 vagy több, a payload törzset tartalmazó `DATA` keretből
  - Opcionálisan egy `HEADERS` keretből és azt követő 0 vagy több `CONTINUATION` keretből

HTTP üzenetek felépítése (2)

- A sorozat utolsó keretében be van kapcsolva az `END_STREAM` jelző
- A `HEADERS` és a `CONTINUATION` keretek között nem fordulhatnak elő más keretek (egyetlen adatfolyamból sem)
- Tilos a `chunked` átviteli kódolás használata
  - Az üzenet payload továbbításához `DATA` kereteket használ a HTTP/2, melyekkel az átvitel történhet darabokban

HTTP üzenetek felépítése (3)



Fejlécmezők

- A fejlécmezők nevét kisbetűssé kell alakítani a HTTP/2 szerinti kódolásuk előtt

Pszeudo-fejlécmezők (1)

- A kérések és válaszok kezdősorának ábrázolásához a HTTP/2 `':'` karakterrel kezdődő pszeudo-fejlécmezőket használ
- Kérés pszeudo-fejlécmezők:
  - **`:method`**: a HTTP metódust tartalmazza
  - **`:scheme`**: a cél URI séma részét tartalmazza
  - **`:authority`**: a cél URI autoritás részét tartalmazza, a `Host` fejlécmező megfelelője
    - Kliensek számára ezt ajánlott használni a `Host` fejlécmező helyett
  - **`:path`**: a cél URI útvonal és lekérdezés részét tartalmazza (szerver-szintű `OPTIONS` metódus esetén `'*'` az értéke)
- Válasz pszeudo-fejlécmezők:
  - **`:status`**: a HTTP állapotkódot tartalmazza (minden válaszban kötelező)

Pszeudo-fejlécmezők (2)

- A pszeudo-fejlécmezők a rendes fejlécmezők előtt kell, hogy szerepeljenek
- A `CONNECT` metódus kivételével minden HTTP/2 kérésben pontosan egy `:method`, `:scheme` és `:path` pszeudo-fejlécmező kötelező
- A HTTP/2 nem teszi lehetővé kérésben a protokoll verzió, válaszban a protokoll verzió és az indok frázis megadását

Példa (1)

```
> GET /index.html HTTP/1.1
> User-Agent: curl/7.47.1
> Host: example.com
> Accept: */*
>
```

```
HEADERS (stream_id=1)
  + END_STREAM
  + END_HEADERS
  :method = GET
  :path = /index.html
  :scheme = https
  :authority = example.com
  user-agent = curl/7.47.1
  accept = */*
```

```
< HTTP/1.1 200 OK
< ETag: "54ef4a17"
< Content-Length: 8192
< Content-Type: text/html
<
< ⟨adatok⟩
```

```
HEADERS (stream_id=1)
  - END_STREAM
  + END_HEADERS
  :status = 200
  etag = "54ef4a17"
  content-length = 8192
  content-type = text/html

DATA (stream_id=1)
  + END_STREAM
  ⟨adatok⟩
```

Példa (2)

```
> GET /index.html HTTP/1.1
> User-Agent: curl/7.47.1
> Host: example.com
> Accept: */*
> If-None-Match: "54ef4a17"
>
```

```
HEADERS (stream_id=1)
  + END_STREAM
  + END_HEADERS
  :method = GET
  :path = /index.html
  :scheme = https
  :authority = example.com
  user-agent = curl/7.47.1
  accept = */*
  if-none-match = "54ef4a17"
```

```
< HTTP/1.1 304 Not Modified
< ETag: "54ef4a17"
<
```

```
HEADERS (stream_id=1)
  + END_STREAM
  + END_HEADERS
  :status = 304
  etag = "54ef4a17"
```

Példa (3)

```
> PUT /image HTTP/1.1
> User-Agent: curl/7.47.1
> Host: example.com
> Accept: */*
> Content-Length: 8192
> Content-Type: image/png
>
> ⟨adatok⟩
```

```
HEADERS (stream_id=1)
  - END_STREAM
  + END_HEADERS
  :method = PUT
  :path = /image
  :scheme = https
  :authority = example.com
  user-agent = curl/7.47.1
  accept = */*
  content-length = 8192
  content-type = image/png

DATA (stream_id=1)
  + END_STREAM
  ⟨adatok⟩
```

Szerver push (1)

- Lehetővé teszi a szervernek a kliens egy kérésével kapcsolatban kéretlen válaszok küldését
  - Olyan válaszok küldése, melyekre a kliensnek szüksége lesz az eredeti kérésre adott válasz teljes feldolgozásához (például stíluslapok, képek)
- A `PUSH_PROMISE` keret révén jelzi a küldő a fogadónak, hogy adatfolyamokat szándékozik létrehozni kéretlen válaszok küldéséhez

Szerver push (2)

- A kliens letilthatja használatát
  - Tilos `PUSH_PROMISE` keret küldése, ha a fogadó végpont `SETTINGS_ENABLE_PUSH` beállításának értéke 0
- A fogadó egy `RST_STREAM` keret küldésével utasíthat vissza egy ígért adatfolyamot

Szerver push (3)

- A szerver egy kéretlen válasz előtt elküldi annak a kérésnek a fejlécmezőit, mely a választ eredményezné
- Egy `PUSH_PROMISE` és azt követő `CONTINUATION` keretek tartalmazzák a kérés fejlécmezőit
  - A szerver számára az ígért válaszokra hivatkozó keretek küldése előtt ajánlott PUSH_PROMISE keretek küldése
  - A `PUSH_PROMISE` keret tartalmaz egy adatfolyam azonosítót, melyet a szerver az ígért válaszhoz foglal le
- Az ígért válaszok mindig a kliens egy explicit kéréséhez tartoznak, a szerver az eredeti kéréshez tartozó adatfolyamon keresztül küldi a `PUSH_PROMISE` kereteket
- A `PUSH_PROMISE` keret küldése után kezdheti meg a szerver az ígért válasz küldését azon az adatfolyamon, melynek azonosítóját a `PUSH_PROMISE` keret tartalmazza

Szerver push (4)

- Példa:

```
HEADERS (stream_id=1)
  + END_STREAM
  + END_HEADERS
  :method = GET
  :path = /index.html
  :scheme = https
  :authority = example.com
  accept = */*
```

```
PUSH_PROMISE (stream_id=1,
  promised stream id=2)
  + END_HEADERS
  :method = GET
  :path = /style.css
  :scheme = https
  :authority = example.com
  accept = */*

HEADERS (stream_id=1)
  - END_STREAM
  + END_HEADERS
  :status = 200
  content-length = 8192
  content-type = text/html

DATA (stream_id=1)
  + END_STREAM
  ⟨adatok⟩
```

Szerver push (5)

- Példa (folytatás):

```
HEADERS (stream_id=2)
  - END_STREAM
  + END_HEADERS
  :status = 200
  content-length = 1024
  content-type = text/css

DATA (stream_id=2)
  + END_STREAM
  ⟨adatok⟩
```

Szerver push (6)

- Kliens oldali támogatás:
  - cURL: igen
  - Hyper: igen
  - Netty: igen
  - nghttp2: igen
- Szerver oldali támogatás:
  - Jetty: igen
  - nghttp2: igen
  - OpenLiteSpeed: nem
  - Undertow: igen
- Gyakorlati megvalósítás:
  - Lásd: Tatsuhiro Tsujikawa, *nghttp2.org Enabled HTTP/2 Server Push*, February 10, 2015. https://nghttp2.org/blog/2015/02/10/nghttp2-dot-org-enabled-http2-server-push/

Kapcsolatkezelés

- A HTTP/2 kapcsolatok perzisztensek
- A kliensek számára nem ajánlott egy adott hoszt egy adott portjához egynél több HTTP/2 kapcsolatot nyitni

HPACK

- A HTTP/2 által használt megoldás fejlécmezők tömörítéséhez

Fogalmak

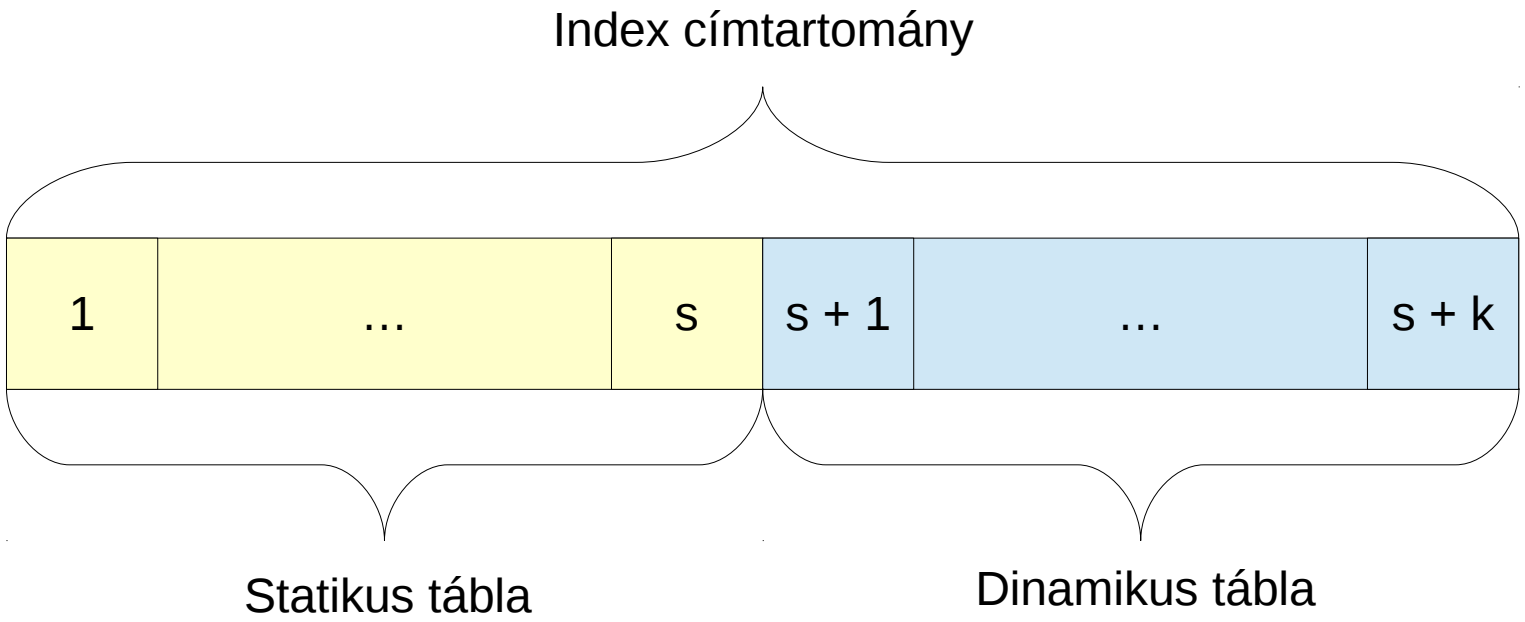
- **Fejlécmező**: egy név-érték pár, ahol a név és az érték egy átlátszatlan oktettsorozat
- **Fejléc lista** (***header list***): együtt kódolt fejlécmezők egy listája, mely ugyanazt a fejlécmezőt többször is tartalmazhatja
- **Fejléc blokk** (***header block***): egy kódolt fejléc lista, mely a továbbításhoz felosztásra kerülhet
- **Fejléc blokk töredék** (***header block fragment***): a fejléc blokk felosztásával nyert okettsorozatok
  - Továbbításuk `HEADERS`, `PUSH_PROMISE` vagy `CONTINUATION` keretekben történik

Indextáblák (1)

- A fejlécmezők kódolása indextáblák használatával történik, melyek a fejlécmezőknek indexeket feleltetnek meg
  - **Statikus tábla**: a gyakran előforduló fejlécmezőkhöz statikusan indexeket hozzárendelő, a specifikációban meghatározott tábla
  - **Dinamikus tábla**: a kódoló és a dekódoló által dinamikusan kezelt, kezdetben üres tábla
    - Olyan FIFO lista, melynek bővítése az elején történik
    - Méretére felső korlát határozható meg

Indextáblák (2)

- A statikus és a dinamikus tábla egyetlen index címtartományt alkot



A statikus indextábla

Index	Név	Érték
1	:authority	
2	:method	GET
3	:method	POST
4	:path	/
5	:path	/index.html
6	:scheme	http
7	:scheme	https
8	:status	200
9	:status	204
…	…	…
58	user-agent	
59	vary	
60	via	
61	www-authenticate	

Fejlécmezők ábrázolása

- Egy fejlécmező kétféle módon ábrázolható a kódolásban:
  - **Indexként**: a statikus vagy dinamikus indextábla egy bejegyzésére hivatkozó indexként
  - **Literálisan**: a fejlécmező nevének és értékének megadásával
    - A fejlécmező neve ábrázolható literálisan vagy valamelyik indextábla egy bejegyzésére hivatkozó indexként
    - A fejlécmező értékének ábrázolása literálisan történik
- Fejlécmező nevének és értékének literális ábrázolása történhet közvetlenül vagy egy statikus Huffman-kóddal

Huffman-kódolás

- A specifikációban meghatározott statikus Huffman-kódok használata

Decimális érték	Karakter	Kód	Hossz (bit)
…	…	…	…
32	' '	|010100	6
33	'!'	|11111110|00	10
34	'"'	|11111110|01	10
35	'#'	|11111111|1010	12
36	'$'	|11111111|11001	13
37	'%'	|010101	6
38	'&'	|11111000	8
39	'''	|11111111|010	11
…	…	…	…

Példa (1)

- HTTP/1.1 kérés mérete: 60 oktett

```
> GET /index.html HTTP/1.1
> Host: example.com
> Accept: */*
>
```

- Ekvivalens HTTP/2 kérés mérete: 9 + 6 + 24 = 39 oktett

```
HEADERS
  + END_STREAM
  + END_HEADERS
  :method = GET
  :scheme = https
  :path = /index.html
  :authority = example.com
  accept = */*
```

Példa (2)

```
< HTTP/1.1 200 OK
< Date: Thu, 12 Mar 2015 15:07:22 GMT
< Last-Modified: Thu, 12 Mar 2015 13:10:35 GMT
< Content-Length: 4096
< Cache-Control: max-age=3600
< Expires: Thu, 12 Mar 2015 16:07:22 GMT
< Content-Type: text/html
<
```

```
HEADERS
  - END_STREAM
  + END_HEADERS
  :status = 200
  date: Thu, 12 Mar 2015 15:07:22 GMT
  last-modified: Thu, 12 Mar 2015 13:10:35 GMT
  content-length: 4096
  cache-control: max-age=3600
  expires: Thu, 12 Mar 2015 16:07:22 GMT
  content-type: text/html
```

- A válasz kezdősor és fejlécmezők mérete: 218 oktett
- Ekvivalens HEADERS keret mérete: 9 + 6 + 98 = 113 oktett

Eltérések az SPDY és a HTTP/2 között

- **Keretek**: eltérő felépítésű keretetek használata
- **Titkosítás**: az SPDY esetén kötelező a TLS használata, a HTTP/2 esetén opcionális
- **Protokoll egyeztetés**: az SPDY egy nem szabványos megoldást használ protokoll egyeztetésre (NPN), a HTTP/2 egy szabványos megoldást (ALPN)
- **Rangsorolás**: a HTTP/2 egy sokkal rugalmasabb rangsorolást használ, mint az SPDY
- **Fejléc tömörítés**: a HTTP/2 egy kifejezetten erre a célra tervezett megoldást használ a fejlécek tömörítéséhez, mely sokkal gyorsabb és memóriahasználat szempontjából sokkal hatékonyabb, mint az SPDY által használt `zlib` tömörítés, bár tömörítési hatásfoka egy kissé rosszabb annál

További ajánlott irodalom

- Daniel Stenberg, *http2 explained*, September 11, 2015. http://daniel.haxx.se/http2/