A Google új, kísérleti QUIC protokolljának teljesítményelemzése

Krämer Zsolt, Megyesi Péter, Molnár Sándor
Nagysebességű Hálózatok Laboratóriuma,
Távközlési és Médiainformatikai Tanszék,
Budapesti Műszaki és Gazdaságtudományi Egyetem,
1117, Magyar tudósok körútja 2.,
Budapest, Magyarország
{kramer,megyesi,molnar}@tmit.bme.hu

***Kivonat*—A webes forgalom átvitelének meghatározó protokollja a 90-es évek óta a HTTP. Az elmúlt 20 évben azonban a weboldalak és webes alkalmazások olyannyira megváltoztak, hogy a protokoll működése a mai Interneten már nem optimális, teljesítményének korlátairól számos publikáció született. A Google az elmúlt években két új protokollt is implementált: 2009-ben a SPDY-t és 2013-ban a QUIC-et. A SPDY fejlesztése során világossá vált, hogy sok esetben a teljesítményt a szállítási rétegben használt TCP protokoll korlátozza. A kísérleti QUIC (Quick UDP Internet Connections) protokoll a hagyományokkal szakítva UDP fölött működik, és mivel a technológia még nagyon friss, ezidáig igen kevés kutatási eredmény került nyilvánosságra a működéséről. Írásunkban összefoglaljuk a SPDY és QUIC protokollok újításait, majd bemutatunk egy átfogó összehasonlító elemzést a QUIC, SPDY és HTTP protokollok teljesítményéről, a weboldalak letöltési idejére koncentrálva. Az eredmények alapján kijelenthető, hogy a legjobb teljesítményű protokollt a hálózat- és a weboldal paraméterei együtt határozzák meg, tehát nem létezik a három közül a körülményektől függetlenül leggyorsabb technológia.**

I. Bevezetés

A mai Internet egyik legszélesebb körben használt protokollja a HTTP (Hypertext Transfer Protocol), mely hírek, videók és megszámlálhatatlan webes alkalmazás átviteléért felel eszközök milliárdjain, az asztali számítógépektől az okostelefonokig. A weboldalaink azonban jelentősen megváltoztak a HTTP/1.1 [1] publikálása óta. Ahogy a webes alkalmazások tovább fejlődnek, az Internet forgalma pedig rohamosan nő, egyre nagyobb jelentőséggel bír az új technológiák kutatása. Az oldalletöltési idő (Page Load Time, PLT) különösen fontos aspektusa a teljesítménynek, melyre a weboldalak korai elhagyásával való kapcsolat [6] is rávilágít.

A Google 2009-ben kezdett bele egy új web transzfer protokoll fejlesztésébe. Ez a SPDY [2], melyet mára számos nagy forgalmú szerveren (Google, Facebook, Twitter) implementáltak, és az összes modern böngésző támogatja. A SPDY a szabványosítás alatt álló HTTP/2 protokoll alapja [4].

Azonban nem csak a HTTP/1.1-nek vannak hátrányai és korlátai. A szállítási rétegben a TCP (Transmission Control Protocol), mely a múltban kimagasló eredményekkel szolgált, szintén problémákkal küzd. A Google-nek lehetősége van hatékonyan elemezni a TCP teljesítményét a mai hálózatokban, mivel az Internet forgalmának 20-25%-a [7] az ő szerverein halad át, a Chrome böngésző pedig több, mint 45%-os piaci részesedéssel a piacvezető webböngésző [8]. Ezekre a tapasztalatokra építve fogott bele a Google a QUIC [3] fejlesztésébe 2013-ban. A QUIC protokoll a SPDY újításait egy új transzport protokoll fölé helyezve mind az alkalmazási-, mind a szállítási rétegben igyekszik áttörést elérni, és ezzel felgyorsítani a Webet.

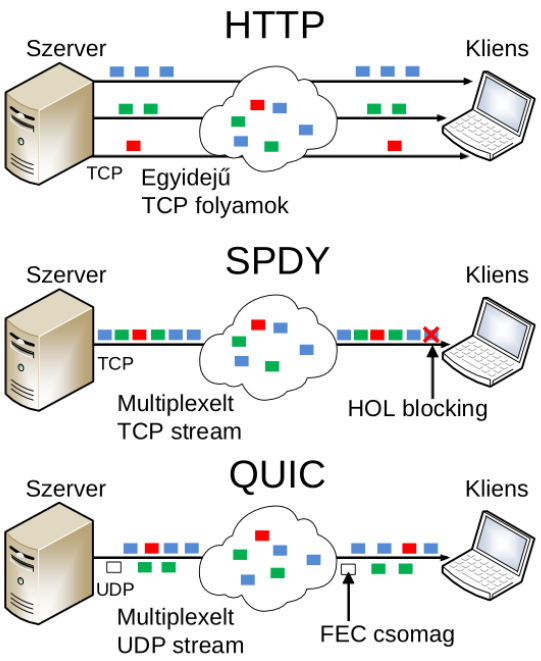
Mivel a QUIC még egy nagyon friss technológia, igen kevés tanulmány vizsgálta eddig. Írásunk elsődleges célja, hogy hozzájáruljon a QUIC protokoll teljesítményének alaposabb megértéséhez, összehasonlítva azt a HTTP/1.1-el és a SPDY-vel.

A II. fejezet összefoglalja a SPDY és a QUIC hátterét, illetve a kapcsolódó irodalmat. A III. fejezetben részletesen bemutatjuk a mérési környezetet, amit a protokollok teljesítményének teszteléséhez használtunk. A IV. fejezet tartalmazza a mérések eredményeit, majd az V. fejezetben összefoglaljuk a tapasztalatokat.

II. A Web protokolljainak evolúciója

II-A. A HTTP/1.1 hiányosságai

A HTTP/1.1 [1] egy kérés-válasz alapú protokoll, melyet a 90-es években terveztek, amikor a weboldalak még jelentősen egyszerűbbek voltak, mint ma. A felhasználói interakciókra ma már közel valós idejű választ várunk egy weboldaltól, melyet a HTTP/1.1 nem képes kiszolgálni. Az egyik legfontosabb, a HTTP teljesítményét hátrányosan befolyásoló mechanizmus a túl sok TCP kapcsolat nyitása a párhuzamosság elérése érdekében. A HTTP folyamok nagy része kis (15KB-nál kisebb), börsztös adatátvitelből áll, több tucat különböző TCP kapcsolaton, ahogy azt az 1. ábra szemlélteti. A TCP azonban hosszú kapcsolatokra optimalizált protokoll, az új kapcsolatok felépítésének költségét pedig nagyban befolyásolja a körülfordulási idő (Round-Trip Time, RTT) nagysága. Amikor a HTTP új TCP kapcsolatok nyitásával igyekszik javítani a teljesítményt, a túl nagy számú TCP kapcsolat torlódáshoz vezet, mely végeredményben rosszabb teljesítményt eredményez. A kapcsolatok száma még tovább nő, amennyiben a webes objektumok különböző domain-ekről érkeznek. A problémára a HTTP pipelining próbált megoldást kínálni, mely azonban nem terjedt el [5].



1. ábra. Adatfolyamok a HTTP, SPDY és QUIC protokollokban

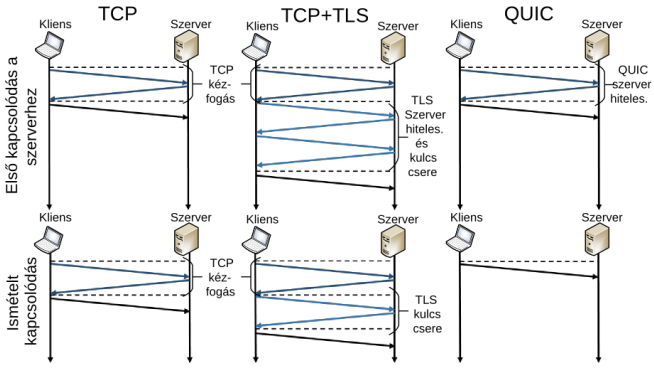
Szintén problémát jelent, hogy a HTTP-ben adatátvitelt kizárólag a kliens kezdeményezhet. Ez különösen beágyazott objektumok letöltésekor jár komoly teljesítménycsökkenéssel. A szervernek ilyenkor minden objektum küldése előtt várnia kell a kliens explicit kérésére, mely csak az után érkezhet meg, hogy a kliens feldolgozta a HTML dokumentumot.

Mivel egy TCP szegmens nem tartalmazhat egynél több HTTP kérést vagy választ, a kliensek jelentős mennyiségű redundáns adatot küldenek TCP SYN csomagok és HTTP fejlécek formájában. Ez a feleslegesen átvitt adatmennyiség különösen naggyá válik a hasznos adathoz képest, amikor sok kis méretű beágyazott objektum található az oldalon. ADSL (Asymmetric Digital Subscriber Line) kapcsolatok esetén, ahol a feltöltési sávszélesség különösen szűkös erőforrás, a redundáns adatok átvitele komolyan megnövelheti a késleltetést. A fejlesztői közösség a múltban az azonos típusú, kis méretű fájlok konkatenálásával igyekezett ezt kivédeni, illetve néhány esetben inline fájlok használatával a HTML dokumentumban. Ezek az eljárások azonban komoly hátrányokkal is rendelkeztek: a fájlok konkatenálása negatív hatással van a cache hatékonyságára, illetve a CSS és JavaScript fájlok konkatenálása késlelteti a feldolgozást [5].

II-B. SPDY

A SPDY[1] protokoll tervezésekor a cél az volt, hogy orvosolja a HTTP említett hiányosságait [2]. A protokoll az alkalmazási rétegben működik, TCP fölött. A SPDY keretező rétege a HTTP-hez hasonlóan kérés-válasz streamekre optimalizált, így azok az alkalmazások, amik HTTP fölött futottak, SPDY

[1]A SPDY “speedy”-ként ejtendő és nem egy mozaikszó



2. ábra. A kapcsolat felépítése a TCP, TLS és QUIC protokollokban

fölött is futhatnak, akár változtatások nélkül. A továbbiakban bemutatjuk a SPDY főbb újításait.

Ahogy az 1. ábrán is látható, a SPDY egy domain-hez egyetlen, multiplexelt TCP kapcsolatot nyit. Az egy kapcsolat (SPDY session) fölött, párhuzamosan kiszolgált kérések száma tetszőlegesen nagy lehet. Ezek a kérések stream-eket, kétirányú adatfolyamokat hoznak létre. Ez a multiplexelés a HTTP pipelining-nál jóval kifinomultabb eljárás a kérések párhuzamosságának biztosítására, mert csökkenti az SSL (Secure Sockets Layer) overhead-et, növeli a szerverek hatékonyságát és segít a torlódáselkerülésben. A stream-ek létrejöhetnek a kliens- vagy a szerveroldalon, és szállíthatnak más streamekkel átlapolt adatot [2].

A SPDY újításai közé tartozik a kérések priorizálhatósága. A kliensnek lehetősége van egy prioritási szintet rendelni minden objektumhoz, majd a szerver ezen prioritásoknak megfelelően ütemezi az objektumok átvitelét. Ez segít elkerülni az olyan eseteket, amikor a csatornán nem kritikus objektumok torlódást okoznak, míg egy magas prioritású kérés (pl egy JavaScript kód modul) várakozni kényszerül.

A protokollban a szerver a kliens explicit kérése nélkül is küldhet adatot a kliensnek (szerver oldali push-mechanizmus). Enélkül a kliensnek először le kell töltenie a HTML dokumentumot és csak ezután kezdeményezheti a másodlagos erőforrások átvitelét. A szerver oldali push-mechanizmus csökkentheti a késleltetést beágyazott objektumok letöltésekor, azonban ronthat a cache hatékonyságán, így a megfelelő optimalizáció kulcsfontosságú.

A SPDY a HTTP fejlécek formájában átvitt redundáns adatmennyiség problémájára is kínál megoldást: a fejléceket egy dedikált stream-ben, tömörített formában viszi át. Egészen a SPDY 3-as verziójáig a protokoll a DEFLATE formátumot használta a redundáns adatok leképezésére, azonban ezzel az eljárással kapcsolatban súlyos sebezhetőségek kerültek napvilágra, mint például a CRIME (Compression Ratio Info-leak Made Easy). Erre válaszul a SPDY 4-es verziójában már a HPACK eljárással kódolják a HTTP fejléceket [9].

A SPDY teljesítményét a mai napig nem értjük megfelelően, ezt bizonyítja, hogy a témában egymásnak ellentmondó kutatási eredmények láttak napvilágot. A Google [12]

és a Microsoft [13] jelentős teljesítményjavulásról számolt be (60% csökkenés a PLT-ben) a HTTP-vel összehasonlítva, ezzel ellentétben az Akamai [14] és a Cable Labs [15] eredményei csupán alacsony mértékű javulást mutatnak, sőt, néhány esetben teljesítményromlást. [16] és [17] szintén kis mértékű teljesítményjavulást mutattak ki nagy körülfordulási idő mellett (például műholdas kapcsolatok esetén), azonban egy másik tanulmány [18] kimutatta, hogy ez 3G hálózatokban nem érvényesül a cellás rendszerek felépítése miatt.

[10] és [11] izolált teszthálózatban vizsgálta a SPDY teljesítményét, nagy kiterjedésű paramétertérben. A két publikáció hasonló eredményeket tartalmaz: i) a SPDY jobban teljesít a HTTP-nél nagy számú objektum vagy nagy RTT esetén, legfőkébb a multiplexelésnek és a tömörített fejléceknek köszönhetően, ii) a SPDY nagyobb teljesítményjavulást ér el alacsony sávszélesség esetén, nagy sávszélesség mellett az eredmények között nincs számottevő különbség, iii) a HTTP jobban teljesít a SPDY-nél nagy csomagvesztés mellett, ahol a SPDY teljesítménye drámaian visszaesik az ú.n. head-of-line blocking (HOL blocking) miatt (bővebben lásd a következő alfejezetet). Ezeket az eredményeket a mi méréseink is megerősítették.

II-C. A TCP korlátai

A SPDY sikeresen túllépett a HTTP/1.1 számos hiányosságán, azonban akadnak a további teljesítményjavulást akadályozó tényezők, melyeket a szállítási rétegben kell keresnünk. A TCP egyik legfontosabb funkciója a torlódásszabályozás. Az évek során számos új TCP verziót javasoltak (például [19], [20]), illetve egyes kutatócsoportok új, alternatív transzport protokollokat is fejlesztettek (például [21]). Sajnos azonban a szállítási réteg protokolljaival, elsősorban a TCP-vel kapcsolatban a tapasztalatok azt mutatják, hogy a protokoll nagyon lassan fejlődik, a fejlesztések pedig még ennél is lassabban terjednek el. Ennek oka, hogy nem csupán szervereken és klienseken kell implementálni, hanem middlebox-ok tömegén szerte az Interneten. Ez azt jelenti, hogy egy újítás a TCP protokollban 10 év-, vagy akár ennél is hosszabb idő alatt terjed el széles körben.

A sávszélesség ma már egyre kevésbé korlátozza a webes teljesítményt (többek közt a lakossági üvegszálas megoldások elterjedése miatt), azonban a késleltetésről gyakran megfeledkezünk. A hálózati RTT a legfontosabb faktor egy új TCP kapcsolat throughput-jában, és értékét nagyban behatárolja a fény terjedési sebessége, így a körülfordulások számának csökkentése az egyetlen út, amivel a késleltetés jelentősen csökkenthető. Ahogy a 2. ábrán látható, a TCP-nek egy körülfordulásra van szüksége, hogy felépítse a kapcsolatot mielőtt a HTTP kérés elküldhető válik. Ha titkosított kapcsolatról beszélünk, akkor ez az idő tovább növekszik; legalább egy RTT-re van szükség a TLS kulcs-cseréhez, az első kapcsolódás esetén pedig még egy körülfordulásra az azonosításhoz.

Fontos még megemlítenünk a head-of-line blocking jelenségét. A TCP megbízható kapcsolatot és sorrendhelyes átvitelt garantál, ez pedig azt jelenti, hogy ha egy csomag elveszik, minden adatküldésnek várnia kell az újraküldésig. [10] és [11] megmutatták, hogy magas csomagvesztés mellett a HTTP jobban teljesít a SPDY-nél, mert a SPDY esetén egy domainhez egyetlen TCP kapcsolat létezik, és a HOL blocking ez esetben az összes stream-et várakozásra kényszeríti, míg a HTTP a több párhuzamosan létező TCP kapcsolat miatt kevésbé érintett.

II-D. QUIC

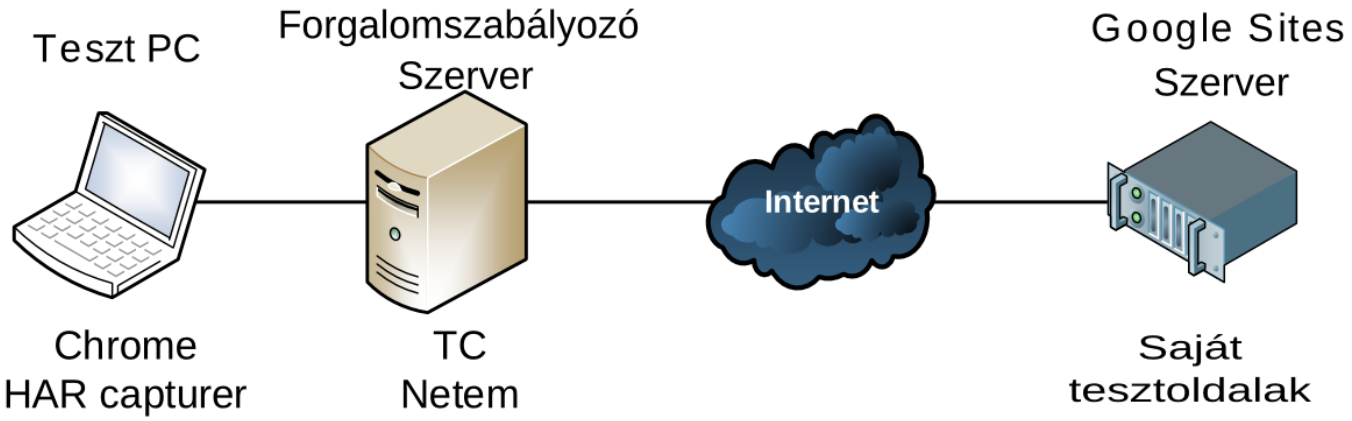
A QUIC protokoll [3] fejlesztésekor az egyik legfontosabb célkitűzés a webes forgalom késleltetésének csökkentése volt. Ehhez szükség volt áttérni TCP-ről UDP-re a transzport rétegben, illetve egy új titkosító protokoll implementálására, mely kiváltja a TLS/SSL-t, így támogatva a kapcsolat felépítéséhez szükséges körülfordulások számának csökkentését, miközben a biztonsága a TLS-sel összemérhető marad [3].

Mivel a QUIC UDP fölött működik, nincs garancia a csomagok sorrendhelyes átvitelére, így elkerülhető a HOL blocking. Egy kliens akár 0-RTT alatt csatlakozhat egy szerverhez, ha korábban már létezett kapcsolat közöttük (lásd a 2. ábrán). Ez úgy érhető el, hogy minden csomag tartalmaz egy, a kapcsolatra vonatkozó azonosítót, mely kiváltja a hagyományos IP fourtuple-t (forrás és cél címek, illetve portok). A plusz körülfordulás a TLS-ben nem követelmény a biztonság szempontjából, kizárólag a kézfogás implementációjából származik. A QUIC-ben implementált titkosító ezen változtat, működése pedig a DTLS-hez (Datagram Transport Layer Security) hasonló [22].

Ahogy az 1. ábrán látható, a QUIC protokoll hibajavító kódolás (FEC) használatával is igyekszik kiküszöbölni a csomagvesztés negatív hatását a teljesítményre. A QUIC hajlandó áldozni a hasznos sávszélességből a késleltetés csökkentéséért a kritikus csomagok (mint például a kapcsolatot kezdeményező UDP csomag) proaktív újraküldésével. A Google mérései alapján a FEC csomagokra fordított 5% extra sávszélesség 8%-kal kevesebb újraküldést eredményez [23].

A torlódásszabályozás megvalósítása a QUIC protokollban logikailag a TCP-CUBIC-kal (illetve opcionálisan a TCP-Reno-val) megegyezően van implementálva. Ezt azonban kiegészíti a packet pacing nevű mechanizmus, mely folyamatos optimalizálás alatt áll. A packet pacing vezérléséhez a QUIC a csomagküldési idők közötti különbségből becsüli meg a rendelkezésre álló sávszélességet, és szabályozza a csomagküldés sebességét. Korai mérések azt mutatták, hogy a packet pacing csökkenti a torlódásból származó csomagvesztések számát [23], azonban a teljesítményt jelentősen ronthatja alacsony csomagvesztési arány mellett [24].

A Google a korai eredményeit [22]-ben és [23]-ben mutatta be, illetve [24] is vizsgálta a QUIC teljesítményét, azonban ezekben az esetekben a metodológia leírása teljesen hiányzik. [25] megállapította, hogy a QUIC magas csomagvesztés esetén jobban teljesít a SPDY-nél, és hogy a FEC modul aktiválása lassítja a QUIC-et. [26] eredményei alapján a QUIC nagy RTT és alacsony sávszélesség mellett képes jobban teljesíteni a SPDY-nél és a HTTP-nél. Az utóbbi két kutatásban a Google publikus QUIC prototípus-szerverét [28] használták, mely a méréseket maximálisan megismételhetővé teszi, ám az



3. ábra. Az összeállított mérési elrendezés

eredmények pontosságát mégis veszélyezteti (bővebben lásd a III. fejezetet).

III. Mérési környezet

Ebben a fejezetben részletesen ismertetjük a metodológiát, aminek segítségével a QUIC, SPDY és HTTP protokollok teljesítményét összehasonlítottuk. A 3. ábra szemlélteti az alkalmazott tesztkörnyezetet. Egy laptopon[2] futtatott Chrome böngészőn a Chrome HAR Capturer [27] segítségével automatizáltuk az oldalletöltéseket, az eredményeket pedig HAR(HTTP ARchive) formátumban mentettük el. A HAR fájlok a letöltési idők vizsgálatához szükséges minden információt tartalmaznak. A Chrome HAR Capturer minden egyes oldalletöltés előtt törli a socket pool-t és a cache-t.

Készítettünk 4 különböző weboldalt, melyeket a Google szerverein (Google Sites) helyeztünk el. Azért ezt a megközelítést választottuk, mert QUIC szervereket eddig kizárólag a Google használ (pl Gmail, YouTube, Google Translate), ezeken kívül pedig csak egy egyszerű szerver modul érhető el, melyen az implementáció tesztelhető [28], azonban a teljesítmény korrekt elemzésére és összehasonlítására nem feltétlenül alkalmas. A méréseinkhez használt egyetemi hálózat és a Google Sites szerver között nagyon alacsony ingadozást tapasztaltunk a sávszélességben és a körülfordulási időben, emiatt úgy véljük, hogy az élő szerveren végzett mérések megfelelő pontosságú eredményekkel szolgálhattak.

A QUIC és a SPDY is multiplexelt kapcsolatot használ, melynek előnye elsősorban olyan weboldalakon jelentkezik, amin nagy számú objektum található. A Google Sites-on elhelyezett weboldalaink kis- (400B-8KB) vagy nagy (128KB) méretű, illetve kis (5) vagy nagy (50) számú objektumot tartalmaztak. Az objektumok képek: a kis méretűek nemzeti zászlók, a nagy méretűek pedig nagy felbontású fényképek[3].

A különböző hálózati körülményeket a netem [30] (része a Linux Traffic Control csomagjának) használatával emuláltuk, így állítottuk be az egyes hálózati konfigurációkat meghatározó sávszélesség, csomagvesztés és késleltetés értékeket. Sávszélességben az alacsony, közepes és magas értékeket 2 Mbps, 10 Mbps, illetve 50 Mbps-ként definiáltuk. A csomagvesztés hatását kétféle beállítás mellett vizsgáltuk: alacsony-, amikor nem adunk a hálózathoz extra csomagvesztést, illetve magas, amikor a TC segítségével mind kimenő-, mind a bejövő csomagok 2%-át véletlenszerűen eldobjuk. A késleltetés esetében az alacsony beállításnál nem adtunk a hálózathoz extra késleltetést, így az átlagos RTT 18 ms volt, míg magas késleltetésnél mindkét irányban 100 ms-os késleltetést emuláltunk, így az átlagos RTT 218 ms lett.

Kategória	Paraméter	Értékek
Hálózati	Sávszélesség	2 Mbps, 10 Mbps, 50 Mbps
	RTT	18 ms, 218 ms
	Csomagvesztés	0%, 2%
Weboldal	Objektumok száma	5, 50
	Objektumok mérete	400 B - 8 KB, 128 KB

I. táblázat. Az egyes szcenáriókat meghatározó paraméterek

Az I. táblázatban látható az öt dimenziós paramétertér összefoglalása. Ez 48 különböző konfigurációt jelent, ahol minden esetben legalább 100 mérést végeztünk.

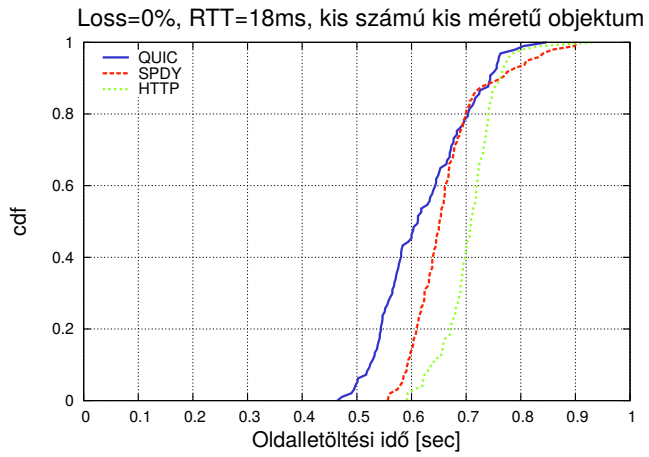
IV. Eredmények

Ebben a fejezetben összehasonlítjuk, hogy a QUIC, SPDY és HTTP protokollok milyen gyorsan tudják letölteni a Google Sites szerveren elhelyezett különböző weboldalakat. A cikk terjedelmi korlátai miatt nem mutatjuk be mind a 48 különböző szcenáriót, ehelyett néhány kiválasztott konfiguráció eredményeire fókuszálunk, melyek érzékeltethetik a főbb tanulságokat.

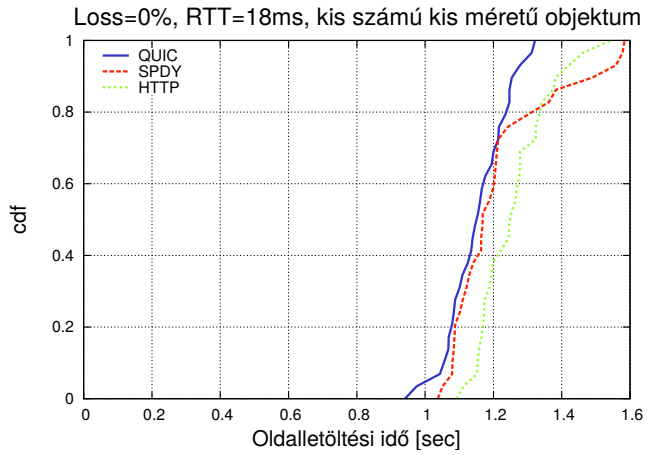
A 4. ábrán látható az oldalletöltési idő (PLT) eloszlásfüggvénye (CDF) alacsony csomagvesztés(loss) és RTT mellett a legkisebb oldalon (kis számú kis méretű objektum). 4a mutatja az 50 Mbps -, 4b pedig a 10 Mbps sávszélességhez tartozó eredményeket. Mindkét esetben csak kis különbség látszik a protokollok teljesítményében. Az átlagok ugyan némileg

[2]Intel Core i5 CPU, 4GB RAM, Ubuntu 14.04 (64bit), Chrome verzió:37.0.2062.94.

[3]A Google Sites automatikusan átméretezi a nagy méretű fotókat 128KB-ra.

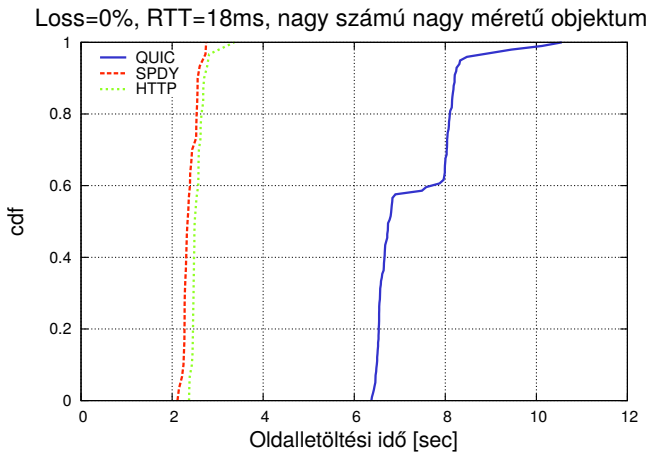


(a) Bw = 50Mbps

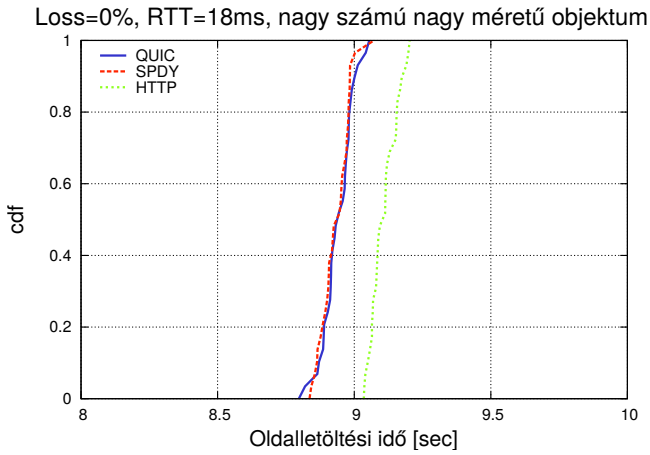


(b) Bw = 10Mbps

4. ábra. Az oldalletöltési idők eloszlása alacsony RTT, alacsony csomagvesztés és kis számú kis méretű objektum esetén



(a) Bw = 50Mbps



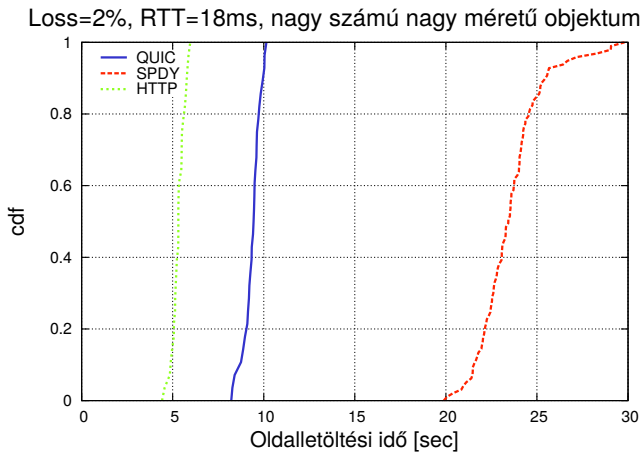
(b) Bw = 10Mbps

5. ábra. Az oldalletöltési idők eloszlása alacsony RTT, alacsony csomagvesztés és nagy számú nagy méretű objektum esetén

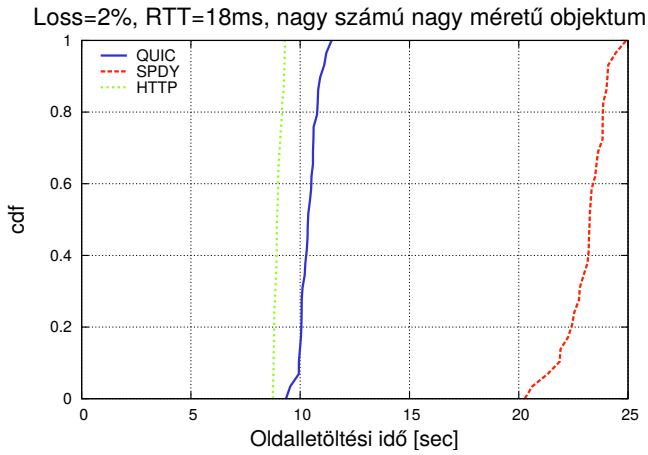
különböznek, de a görbék szórása nagyobb ennél a különbségnél, így nem jelenthetjük ki, hogy bármelyik protokoll egyértelműen gyorsabb lenne a másik kettőnél. Az események nagyon hasonlóak voltak a nagy számú kis méretű objektumot tartalmazó -, és a kis számú nagy méretű objektumot tartalmazó weboldalon. Megállapíthatjuk tehát, hogy a protokollok nagyjából egyformán teljesítenek jó hálózati körülmények között (nagy sávszélesség, alacsony csomagvesztés, alacsony RTT) kis-, vagy közepes méretű oldalak letöltésekor.

Jóval nagyobb különbség mutatkozik a protokollok teljesítményében nagy méretű weboldalak esetén. A 5. ábra alacsony csomagvesztés és RTT mellett ábrázolja az értékeket a nagy számú nagy méretű objektumot tartalmazó oldal letöltésekor. Amikor a sávszélességet 10 Mbps-ra állítottuk (5b), a letöltési idők ismét hasonlóak lettek, azonban az 50 Mbps-os szcenárióban (5a) a QUIC jelentősen rosszabbul teljesít a SPDY-nél és a HTTP-nél. Ebben az esetben a QUIC átlagos oldalletöltési ideje több, mint háromszorosa a másik két protokollénak. Ennek fő oka a QUIC packet pacing mechanizmusa: a QUIC goodput-ja nem képes elérni egy ilyen link kapacitását. Mivel ez a viselkedés sem a 10 Mbps-os konfigurációnál, sem a kisebb méretű oldalak letöltésekor nem jelentkezett, arra következtethetünk, hogy a QUIC teljesítménye csak nagy rendelkezésre álló sávszélesség, és nagy letöltendő adatmennyiség mellett csökken drasztikusan a másik két protokollhoz képest.

A 6. ábra és a 5. ábra konfigurációja kizárólag a magas csomagvesztésben különbözik. Ebben az esetben a SPDY nagyon rosszul teljesít: az átlagos PLT többszörösére nőtt a 2%-os csomagvesztés hozzáadása után. A HOL blocking drámai negatív hatását a SPDY teljesítményére korábbi írások is bemutatták, mint [10], [11], ekkora teljesítménycsökkenést azonban egyik kutatás sem mutatott ki, mert a szerzők nem vizsgálták a csomagvesztés hatását ilyen magas sávszélesség mellett. Az eredményekből az is látszik, hogy QUIC-et használva az átlagos PLT az alacsony csomagvesztésű esethez képest csak 20%-kal nőtt, míg a HTTP-nél nagyjából megduplázódott. Ez elsősorban a QUIC gyorsabb csomagvesztés utáni helyreállási mechanizmusaival magyarázható, másodsorban pedig a FEC használatával. A 7. ábrán alacsony sávszélesség és magas RTT

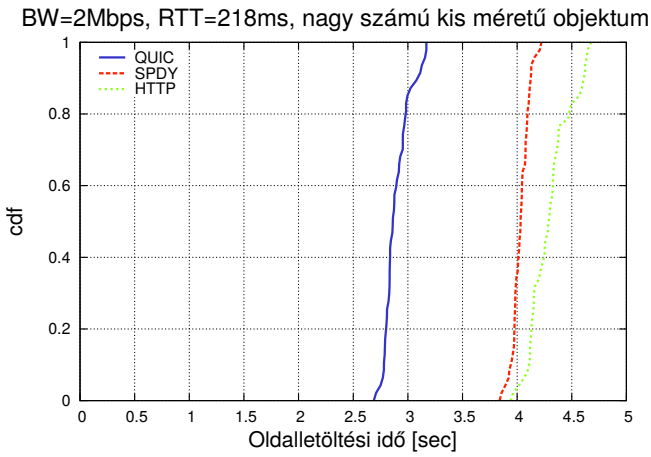


(a) Bw = 50Mbps

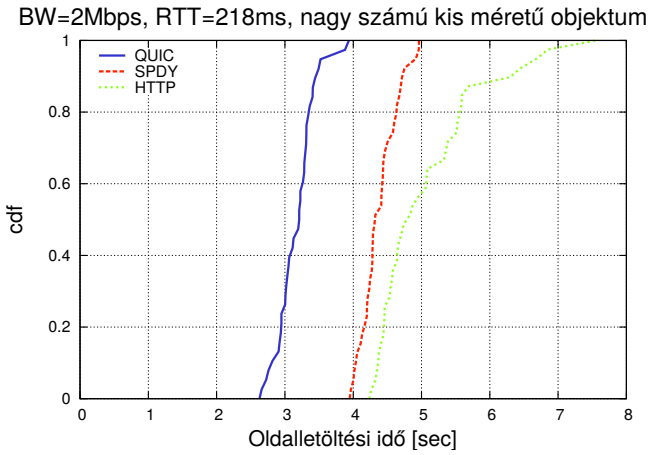


(b) Bw = 10Mbps

6. ábra. Az oldalletöltési idők eloszlása alacsony RTT, magas csomagvesztés és nagy számú kis méretű objektum esetén



(a) Alacsony csomagvesztés



(b) 2% csomagvesztés

7. ábra. Az oldalletöltési idők eloszlása 2 Mbps sávszélesség, magas RTT és nagy számú kis méretű objektum esetén

mellett láthatóak a nagy számú kis méretű objektumot tartalmazó oldal letöltési idői veszteségmentes (7a) és veszteséges (7b) környezetben. Az eredményeink igazolják a feltételezést, miszerint a SPDY és QUIC multiplexelt kapcsolatai komoly előnyt jelentenek sok kis méretű objektum letöltése esetén. A QUIC 0-RTT kapcsolódási ideje is komoly jelentőséggel bírt: a QUIC messze a leggyorsabb protokoll, 25-30%-kal megelőzve a SPDY-t és 35-40%-kal a HTTP-t. Fontos még rámutatni arra, hogy a SPDY és a HTTP teljesítménye között a különbség 7-12%, ami megerősíti azokat az állításokat a SPDY irodalmában, melyek szerint a protokoll csak kis mértékben gyorsabb a HTTP-nél.

A mérési eredményeinket a 8. ábrán foglaljuk össze, egy döntési fa formájában. Egy protokollt akkor tekintünk jobbnak egy másiknál egy adott szcenárióban, ha az esetek legalább 70%-ában legalább 10%-kal alacsonyabb PLT-vel rendelkezik. Amennyiben két protokoll gyorsabbnak bizonyult a harmadiknál, de egymásnál nem, a fa megfelelő levelén mindkettőt feltüntettük. Azokban az esetekben, ahol sem a leggyorsabb, sem a leglassabb protokollt nem lehetett a fenti kritérium alapján meghatározni, a levelet egyenlő eredményként jelöltük meg. A döntési fa és a teljes mérési adatbázis alapján az alábbi következtetéseket vonhatjuk le:

-A QUIC rosszul teljesít, amikor magas sávszélesség mellett nagy mennyiségű adatot kell letölteni

-Másfelől, a QUIC remekül teljesít a másik két protokollal összehasonlítva magas RTT mellett, különösen, ha a sávszélesség alacsony

-A magas csomagvesztés kevésbé rontja a QUIC teljesítményét, mint a másik két protokollét

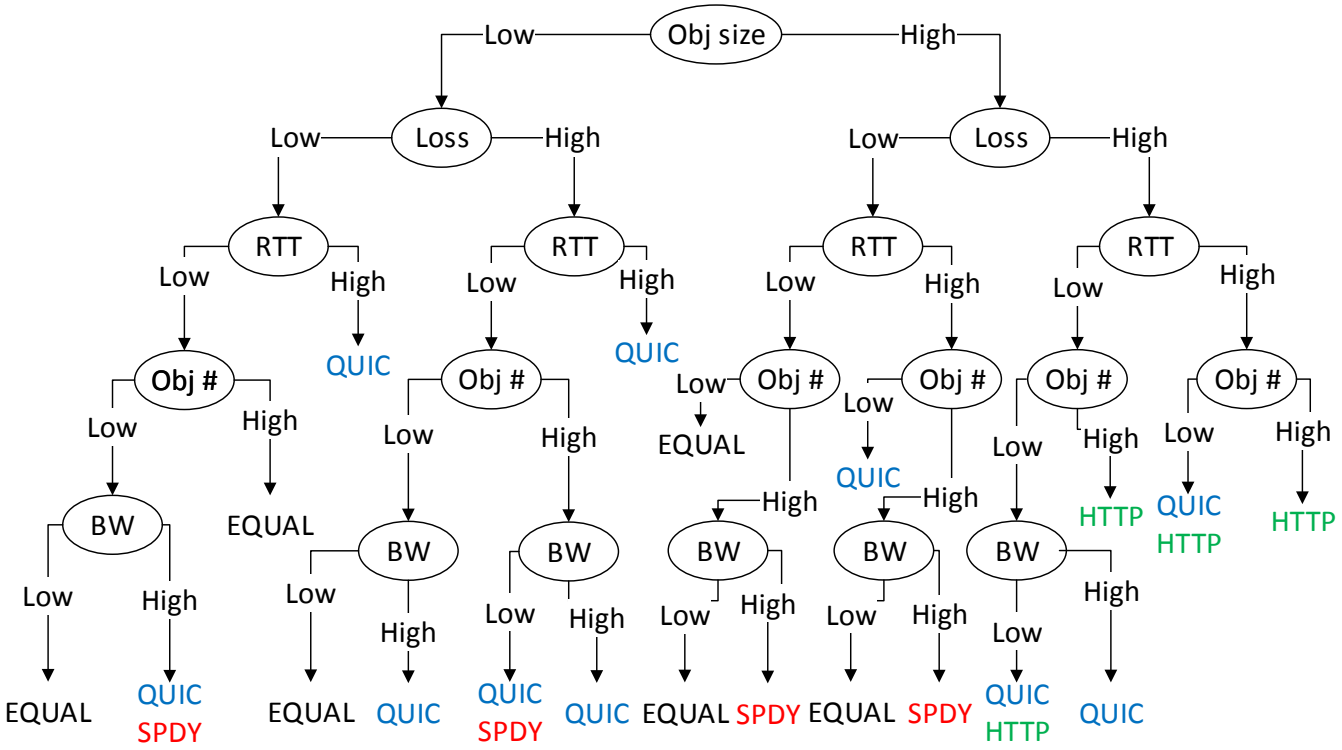
-A SPDY teljesítménye nagyon érzékeny a csomagvesztésre a HOL blocking miatt

-A kis méretű objektumok átvitelekor a multiplexelt kapcsolat előnyt jelent

-Nagy sávszélesség, magas csomagvesztés, és nagy számú nagy méretű objektum esetén a HTTP a leggyorsabb protokoll

V. Konklúzió

A QUIC (Quick UDP Internet Connections) egy új protokoll, melyet a Google 2013 óta fejleszt, célja pedig a SPDY-



8. ábra. Döntési fa, mely hozzárendeli a leggyorsabb protokollt a paramétertér adott pontjához

hez képest további nyereség elérése a szállítási rétegben, ezzel a webes forgalom gyorsabb átvitele. A TCP használata helyett a protokollt UDP fölé implementálták. Ebben a tanulmányban bemutattunk egy összehasonlító elemzést a QUIC, SPDY és HTTP protokollok teljesítményéről, és beazonosítottuk az ezt leginkább meghatározó környezeti paramétereket. Építettünk egy egyszerű tesztkörnyezetet, melyben egy laptopot használva töltöttünk le különböző weboldalakat a Google Sites szerverről, és egy shaper szerver használatával különböző hálózati körülményeket emuláltunk.

Az esetek több, mint 40%-ában a kísérleti QUIC protokoll jelentősen javított a letöltési időkön, de az eredményeink azt is megmutatják, hogy a HTTP képes jobban teljesíteni a multiplexelt protokolloknál nagyméretű objektumok letöltésekor. Az eredmények alátámasztják a korábbi publikációk ([10], [11]) állítását, miszerint a SPDY teljesítményére nagyon erős negatív hatással van a csomagvesztés.

Nagy RTT értékek mellett a QUIC kiemelkedően jól teljesített. Mivel a mobil Internet kapcsolatok (különösen a 3G) esetén a késleltetés általában magas, amikor a QUIC protokoll kilép a kísérleti állapotból, javasolt az implementálása azokon a webszervereken, melyek nagy mobil forgalmat bonyolítanak. A protokoll szintén alapértelmezetten használható lehet a Chrome böngészőben Android platformon.

Az egyetlen beazonosított körülmény, mely a QUIC teljesítményét negatívan befolyásolta a másik két protokollhoz képest, a nagy sávszélesség volt. Ennek egyik magyarázata a packet pacing mechanizmus lehet, mely megakadályozza, hogy a protokoll kihasználhassa egy nagysebességű link kapacitását. Szintén problémát okozhat, ha a hálózatüzemeltetők biztonsági megfontolásokból korlátozzák az UDP forgalmat [31]. A QUIC fölötti webes forgalom, és a veszélyesnek ítélt UDP forgalmak hatékony megkülönböztetésének kidolgozása a fejlesztők és a hálózatüzemeltetők közös feladata a közeljövőben.

Terveink között szerepel a vizsgálatok folytatása, és a QUIC protokoll fejlődésének nyomon követése.

Köszönetnyilvánítás

A szerzők szeretnék megköszönni Szabó Géza és Rácz Sándor (TrafficLab, Ericsson Research Hungary) ötleteit és hasznos megjegyzéseit a cikkel kapcsolatban.

Hivatkozások

[1] Hypertext Transfer Protocol (HTTP/1.1) - RFC 2616. Letöltve: 2015.09.04. Elérhető online: http://tools.ietf.org/html/rfc2616
[2] SPDY Protocol - Draft 3. Letöltve: 2015.09.04. Elérhető online: http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft3
[3] QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2. Letöltve: 2015.09.04. Elérhető online: http://tools.ietf.org/html/draft-tsvwg-quic-protocol-01
[4] Hypertext Transfer Protocol Version 2 (HTTP/2) - RFC 7540. Letöltve: 2015.09.04. Elérhető online: https://tools.ietf.org/html/rfc7540
[5] Grigorik, I.: Making the web faster with HTTP 2.0. Communications of the ACM, Vol. 56, No. 12, pp. 42–49 (2013)
[6] Diriye, A., White, R., Buscher, G., Dumais, S.: Leaving so soon?: understanding and predicting web search abandonment rationales. In Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM '12), New York, NY, USA (2012)

[7] Sandvine - Global Internet Phenomena. Letöltve: 2015.09.04. Elérhető online: https://www.sandvine.com/trends/global-internet-phenomena/

[8] Market Share of Web Browsers (September 2014). Letöltve: 2015.09.04. Elérhető online: http://www.w3counter.com/globalstats.php?year=2015&month=8

[9] HPACK: Header Compression for HTTP/2 - RFC 7541. Letöltve: 2015.09.04. Elérhető online: https://tools.ietf.org/html/rfc7541

[10] Wang, X. S., Balasubramanian, A., Krishnamurthy, A., Wetherall, D.: How Speedy is SPDY? In Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, Seattle, WA, USA (2014)

[11] Elkhatib, Y., Tyson, G., Welzl M.: Can SPDY Really Make the Web Faster? In Proceedings of IFIP Networking Conference, Trondheim, Norway (2014)

[12] SPDY: An experimental protocol for a faster web. White Paper. Letöltve: 2015.09.04. Elérhető online: http://www.chromium.org/spdy/spdy-whitepaper

[13] Padhyeand, J., Nielsen, H. F.: A Comparison of SPDY and HTTP Performance. Microsoft Technical Repor MSR-TR-2012-102.

[14] Podjarny, G.: Not as SPDY as You Thought. Letöltve: 2015.09.04. Elérhető online: http://www.guypo.com/technical/not-as-spdy-as-you-thought/

[15] White, G., Mule, J. F., Rice, D.: Analysis of SPDY and TCP Initcwnd. White Paper. Letöltve: 2015.09.04. Elérhető online: http://tools.ietf.org/html/draft-white-httpbis-spdy-analysis-00

[16] Cardaci, A., Caviglione, L., Gotta, A., Tonellotto, N.: Performance Evaluation of SPDY Over High Latency Satellite Channels. In Personal Satellite Services, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Vol. 123, pp. 123–134 (2013)

[17] Wang, X. S., Balasubramanian, A., Krishnamurthy, A., Wetherall D.: Demystifying page load performance with WProf. In Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI '13), Lombard, IL, USA (2013)

[18] Erman, J., Gopalakrishnan, V., Jana, R., Ramakrishnan, K. K.:Towards a SPDY'ier mobile web? In Proceedings of the 9th International Conference on emerging Networking EXperiments and Technologies (CoNEXT'13), Santa Barbara, CA, USA (2013)

[19] Afanasyev, A., Tilley, N., Reiher, P., Kleinrock, L.: Host-to-Host Congestion Control for TCP. In IEEE Communications Surveys and Tutorials, Vol. 12, No. 3, pp. 304–342 (2010)

[20] Molnár, S., Sonkoly, B., Trinh, T. A.: A Comprehensive TCP Fairness Analysis in High Speed Networks. Computer Communications, Elsevier, Vol. 32, No. 13-14, pp. 1460–1484 (2009)

[21] Molnár, S., Móczár, Z., Sonkoly, B.: How to Transfer Flows Efficiently via the Internet? In Proceedings of International Conference on Computing, Networking and Communications (ICNC 2014), Honolulu, USA (2014)

[22] QUIC: Design Document and Specification Rationale. Letöltve: 2015.09.04. Elérhető online: https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/edit

[23] Roskind, J.: QUIC - Multiplexed Stream Transport over UDP. IETF-88 TSV Area Presentation. Letöltve: 2015.09.04. Elérhető online: http://www.ietf.org/proceedings/88/slides/slides-88-tsvarea-10.pdf

[24] Taking Google's QUIC For a Test Drive. Letöltve: 2015.09.04. Elérhető online: http://www.connectify.me/taking-google-quic-for-a-test-drive/

[25] Carlucci, G., De Cicco, L., Mascolo, S.: HTTP over UDP: an Experimental Investigation of QUIC. In Proceedings of The 30th ACM/SIGAPP Symposium On Applied Computing (SAC 2015), Salamanca, Spain (2015)

[26] Das, R. S.: Evaluation of QUIC on Web Page Performance. Master's Thesis, Massachusetts Institute of Technology, MA, USA, 2014.

[27] Chrome HAR Capturer. Letöltve: 2015.09.04. Elérhető online: https://github.com/cyrus-and/chrome-har-capturer

[28] QUIC Test Server. Letöltve: 2015.09.04. Elérhető online: http://www.chromium.org/quic/playing-with-quic

[29] Apache SPDY Module. Letöltve: 2015.09.04. Elérhető online: https://code.google.com/p/mod-spdy/

[30] Hemminger, S.: Network Emulation with NetEm. In Proceedings of the 6th Australia's National Linux Conference (LCA2005), Canberra, Australia (2005)

[31] Byrne, C.: Advisory Guidelines for UDP Deployment. Letöltve: 2015.09.04. Elérhető online: https://tools.ietf.org/html/draft-byrne-opsec-udp-advisory-00