Aalto-yliopisto
Sähkötekniikan korkeakoulu
Tietoliikennetekniikan koulutusohjelma

Pyry Aaltonen

Viiveiden vähentäminen WWW-liikenteessä

Diplomityö
Espoo, 10. heinäkuuta 2016

Valvoja: Dos. Kalevi Kilkki
Ohjaaja: DI Pasi Sarolahti

Aalto-yliopisto
Sähkötekniikan korkeakoulu
Tietoliikennetekniikan koulutusohjelma

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä:	Pyry Aaltonen		
Työn nimi:	Viiveiden vähentäminen WWW-liikenteessä		
Päiväys:	10. heinäkuuta 2016	**Sivumäärä:**	58
Pääaine:	Tietoverkot	**Koodi:**	S-902
Valvoja:	Dos. Kalevi Kilkki		
Ohjaaja:	DI Pasi Sarolahti		

Työssä esitellään WWW-liikenteen nykytila ja tutustutaan äskettäin esiteltyihin protokolliin, joiden tarkoituksena on erityisesti vähentään viiveitä WWW-liikenteessä. Tutkimuksessa vertaillaan uusien protokollien ominaisuuksia nykyisiin protokolliin. Nykyisten protokollien käsittelyssä painotetaan WWW-liikenteen kannalta merkittäviä ominaisuuksia ja kipukohtia. Tutkimus paneutuu WWW-liikenteen kannalta olennaisiin uudistuksiin ja muutoksiin, joita uudet protokollat sisältävät. Uusien protokollien ominaisuuksia analysoidaan ja niiden soveltuvuutta nykyiseen WWW-liikenteeseen arvioidaan.

Sovelluskerroksen protokollan yleistyminen on tapahtunut työn kirjoituksen aikana, ja työssä todetaan yleistymisen jatkuvan tulevaisuudessa. Tutkimus osoittaa sovelluskerrokselle tarjotun uuden protokollan olevan tarpeellinen uudistus. Siirtokerrokselle tarkoitettu uusi protokolla on vielä joiltain osin keskeneräinen, mutta kehityksenalainen. Siirtokerroksen protokolla tarjoaa useita merkittäviä etuja verrattuna nykyisiin siirtoprotokolliin. Tutkimus osoittaa, että siirtokerroksen ja sovelluskerroksen protokollien suunnitellut synergiat ovat eduksi molempien protokollien toiminnan kannalta. Siirtokerroksen protokolla on kohdistetusti optimoitu tietylle sovelluskerroksen protokollalle. Lisäksi tehdään huomioita, joiden perusteella todetaan, ettei uusi siirtokerroksen protokolla tule täysin syrjäyttämään nykyistä protokollaa. Uuden protokollan todetaan olevan varteenotettava vaihtoehto nykyiselle siirtokerroksen protokollalle.

Asiasanat:	TCP, IP, HTTP, HTTP/2, QUIC, SPDY, TCP/IP, UDP, WWW, viiveet
Kieli:	Suomi

Aalto University
School of Electrical Engineering
Degree Programme in Communications and Networking

ABSTRACT OF
MASTER'S THESIS

Author:	Pyry Aaltonen		
Title:	Reducing delays in WWW-traffic		
Date:	July 10, 2016	**Pages:**	58
Major:	Data Communication Networks	**Code:**	S-902
Supervisor:	Dos. Kalevi Kilkki		
Advisor:	DI Pasi Sarolahti		

This thesis presents the current state of web traffic and introduces new protocols, which are designed to reduce delays in web traffic. The study compares the features of the new protocols to existing protocols. The introduction of existing protocols focuses on relevant features and drawbacks of web traffic. The study focuses on the reforms and modifications in the web traffic related protocols. The new protocol features are analysed and their applicability is evaluated to the current web traffic.

The proliferation of application-layer protocol has occurred during the writing process of this thesis. It is concluded that the proliferation will continue in the future. The research refers that the application-layer protocol is a necessary reform. The transport-layer protocol is still incomplete in some respects, but it is shown to be in a constant development process. The transport-layer protocol provides a number of significant advantages over existing transport-layer protocols. The transport-layer and application-layer protocols have been designed concurrently. The research evaluates that the concurrent design is beneficial for both protocols. The transport-layer protocols optimisation is specified for the application-layer protocol. Furthermore, this thesis indicates that the new transport-layer protocol will not completely replace the current protocol. The new protocol is found to be a viable alternative to the current transport-layer protocol.

Keywords:	TCP, IP, HTTP, HTTP/2, QUIC, SPDY, TCP/IP, UDP, WWW, delays
Language:	Finnish

Alkusanat

Tämä diplomityö on tehty opinnäytteeksi Aalto-yliopiston Sähkötekniikan korkeakoululle. Työn valvojana on toiminut dosentti Kalevi Kilkki ja ohjaajana diplomi-insinööri, Pasi Sarolahti. Kiitokset heille ohjauksesta, neuvoista ja kannustavasta asenteesta.

Kiitos opiskelukavereille, yhdessä puurtaminen ja kirittely loppusuoralla on ollut antoisaa, niin diplomityön kuin muidenkin kurssien kohdalla. Kiitos perheelle kannustuksesta ja sopivasta painostuksesta, ystäville tuesta ja Katariinalle kärsivällisyydestä, jaksamisesta ja myötäelämisestä.

Espoo, 10. heinäkuuta 2016

Pyry Aaltonen

Lyhenteet ja symbolit

AEAD	Authenticated Encryption and Associated Data
DNS	Domain Name System
FEC	Forward Error Correction
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTP/2	Hypertext Transfer Protocol version 2
IP	Internet Protocol
IETF	Internet Engineering Task Force
IoT	Internet of Things
MTU	Maximum Transmission Unit
QUIC	Quick UDP Internet Connection
RFC	Request for Comments
RTT	Round-trip Time
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URL	Uniform Resource Locator
WLAN	Wireless Local Area Network
WWW	World Wide Web

Sisältö

Luku 1

Johdanto

Internetin käyttötarkoitus ja käyttömenetelmät ovat internetin historian aikana muuttuneet merkittävästi. Verkkoliikenteessä on käytössä useita protokollia, jotka ovat säilyneet lähes alkuperäisessä muodossaan. Internet on kehittynyt ja muuttunut verkkoliikenteen protokollien elinkaaren aikana valtavasti. Mobiililaitteiden tulo verkkoon on muuttanut tapaa käyttää verkkoa. Internetin alkuajoilta muutos staattisista sivuista interaktiivisiin sivustoihin on ollut valtava. Verkkoa käytetään entistä enemmän erilaisten sosiaalisten yhteyksien ylläpitoon, ihmisten väliseen kanssakäymiseen ja monenlaisten videopalvelujen käyttöön.

Erityisesti videopalveluiden lisääntynyt käyttö on koetellut perinteisten verkkoprotokollien suorituskykyä. Suoratoistovideopalvelut ovat lisänneet tasaisesti suosiotaan 2000-luvulla. Niiden käyttämä tiedon määrä on moninkertaista verrattuna perinteisiin tekstipohjaisiin verkkosivustoihin. Erityispiirteenä videotoistossa on liikenteen kriittisyys viiveille, sillä liikkuva kuva edellyttää, että seuraava kuvakehys on valmiina ilman erillistä odottelua. Viiveet ovat korostuneet verkkosivustojen kasvaessa ja sirpaloituessa. Käytössä olevat vanhat protokollat ei sellaisenaan ole pystyneet vastaamaan tyydyttävästi verkkoliikenteen muutokseen. Viiveiden tulisi pienentyä verkkojen kapasiteetin kasvaessa. Protokollien rakenteelliset ongelmat ovat kuitenkin rajoittaneet viiveiden pienentymistä, vaikka linkkien kapasiteetti onkin kasvanut. Nykyisten verkkoprotollien muutospaine on kasvanut, mutta tähän saakka on tyydytty yrityksiin parantaa nykyisten protokollien toimintaa. Muutoksen tekeminen nykyisiin verkkoprotokolliin on työlästä ja hankalaa, koska yhteiskunnan ja yrityksien monet kriittisetkin toiminnot ovat nykyisin verkossa. WWW-liikenteessä käytettyjen protokollien, varsinkin siirtokerroksella käytetyn protokollan, uudistaminen on vaativaa, sillä kuten aiemmin to-

dettua on verkkoliikenne kasvanut alkuajoistaan niin räjähdysmäisesti ettei se ole kaikilta osin ollut täysin hallitua. Olemassa olevien protokollien päälle on rakennettu valtavia määriä ohjelmistoja ja laitteita, jotka nojaavat siihen ajatukseen, että käytössä olevat protokollat olisivat muuttumattomia.

Protokollien päivityksessä on pyritty huomioimaan käytössä olevan tekniikan vaatimukset, ja rakentamaan protokollapino, joka on joustava olemassa olevien sovellusten kanssa. Uusien protokollien selkeänä tavoitteena on ollut yhteensopivuus olemassa olevan tekniikan kanssa. Moni asia protokollissa on muuttunut valtavasti, mutta muutos on pyritty tekemään niin, että se aiheuttaisi mahdollisimman pienen muutoksen olemassa oleviin sovelluksiin, kuten välityspalvelimiin ja selaimiin.

Tässä työssä analysoidaan nykyisiä WWW-liikenteen kannalta oleellisia HTTP/1.1 ja TCP-protokollia. Työssä paneudutaan HTTP/1.1 ja TCP-protokollien rajoitteisiin painottaen viiveiden kannalta oleellisia ominaisuuksia. Lisäksi työssä kuvataan HTTP/2.0 ja QUIC -protokollia ja niiden ehdottamia parannuksia.

1.1 Työn tavoitteet ja rakenne

WWW-liikenteessä käytettyjä protokollia käsitellään työn alussa. Esiteltävät protokollat ovat HTTP-protokolla ja TCP-protokolla. HTTP-protokolla on sovelluskerroksen protokolla, joka on tarkoitettu siirtämään tiedostoja, jotka kuvaavat WWW-sivustojen sisältöä. TCP-protokolla on siirtokerroksen protokolla, joka vastaa ylempien kerrosten tiedon sovittamisesta sopivaksi jotta se voidaan kuljettaa IP-protokollalla kohteeseensa. Työssä keskitytään HTTP- ja TCP-protokolliin, sillä nimenomaan niiden tehtävänä oleva yhteyksien avaaminen, käsittely, ja niissä mahdollisesti muodostuvat viiveet, ovat merkittäviä asiakaskokemuksen kannalta. Yhteyksien avaamisessa ja käsittelyssä syntyvät viiveet aiheuttavat asiakkaan päässä tarpeetonta odottelua. Viiveet kuormittavat osaltaan palvelinresursseja. Esimerkiksi välimuistien ja erilaisten puskurien täyttyminen aiheuttaa asiakkaille näkyvää viivettä. Palvelinresurssien loppuminen aiheuttaa pahimmassa tapauksessa palvelun saavuttamattomuuden.

Protokollat esitellään yleisellä tasolla. Työssä käydään läpi niiden toimintaperiaatteet ja rakenne. Protokollien esittelyn jälkeen työssä paneudutaan protokollien ominaisuuksiin tarkemmin. Käsittelyssä pyritään paneutumaan protokollien niihin ominaisuuksiin, jotka tuottavat viivettä WWW-liikenteelle. Tarkoituksena on alustaa lukijalle ymmärrys WWW-liikenteen nykytilas-

ta, jotta tutkimusosassa esiteltävät uudet protokollat ja niiden ominaisuudet saavat vertailukohdan.

Työssä esitellään kaksi uutta protokollaa, HTTP/2.0 ja QUIC. HTTP/2.0-protokolla on sovelluskerroksen protokolla ja se on tarkoitettu korvaamaan sovelluskerroksella nykyisin operoiva HTTP/1.1-protokolla. QUIC on siirtokerroksen ja sovelluskerroksen väliin esitelty protokolla, joka on tarkoitettu vaihtoehdoksi siirtokerroksen TCP-protokollalle. QUIC-protokolla on suunniteltu erityisesti WWW-liikenteen tarpeita ajatellen. QUIC-protokolla pyrkii tarjomaan TCP-protokollan ominaisuudet ja lisäämään sen päälle toimivamman ja viiveettömämmän toimintaympäristön. Protokollat esitellään kattavasti, tarkoituksena on luoda lukijalle selkeä kuva tarjolla olevista uutuuksista.

Esittelyjen jälkeen työssä analysoidaan protokollien ominaisuuksia. Protokollien analysoinnissa keskitytään erityisesti niihin ominaisuuksiin, jotka voisivat olla tärkeitä viiveiden kannalta. Tälläisiä ominaisuuksia ovat esimerkiksi yhteydenavaus, -ylläpito ja protokollien otsakkeet. Työssä verrataan uusien protokollien esiteltyjä ominaisuuksia vanhojen protkollien vastaaviin tarjolla oleviin ominaisuuksiin. Lisäksi työssä analysoidaan uusien protokollien tuomien uudistusten todellisia mahdollisuuksia korvata olemassa olevat protokollat ja tarjota viiveettömämpää WWW-palvelua.

Työn tavoitteena on tarjota uusi tutkimus tulevista protokollista, ja arvioida niiden ominaisuuksia objektiivisesti. Arviointi toteutetaan tarkastelemalla protokollakuvauksia. Työssä vertaillaan uusien protokollien ominaisuuksia ja mahdollisuuksia nykyisten protokollien haastajiksi. Tavoitteena on tehdä mahdollisimman tarkka tutkimus uusien protokollien tarjoamista mahdollisuuksista WWW-liikenteelle.

Luku 2

Hypertext Transfer Protocol

HTTP-protokollan arkkitehtuurinen rakenne ja toimintaperiaatteet ovat pysyneet käytännössä muuttumattomina vuodesta 1996 lähtien, jolloin julkaistiin HTTP:n 1.0 versio. HTTP-protokollaan on tullut päivityksiä ja uudistuksia vuoden 1996 jälkeenkin, mutta käytännössä vain kaksi suurempaa päivitystä. Suuremmat päivitykset on julkaistu vuosina 1997 ja 1999. Päivitysten versionumero on molemmissa päivityksissä HTTP/1.1, mutta ne on erottu 1.1a ja 1.1b versioiksi. Päivitetyt versiot ovat korvanneet aiemman version, eli käytännössä versioita ei ole käytetty rinnakkain kuin siirtymäajan. [4, 10, 11]

HTTP-protokolla on Internetin sovelluskerroksen protokolla. HTTP-protokollan tarkoituksena on mahdollistaa hypertekstin siirtäminen ja vaihtaminen palvelinten ja asiakkaiden välillä. Hyperteksti on tekstimuoto, jota on yleisesti käytetty tietokonejärjestelmissä. Hyperteksti eroaa tavallisesta tekstistä sen sisältämien hyperlinkkien osalta. Hyperlinkit ovat käytännössä viittauksia tekstien välillä. Hyperlinkit mahdollistavat välittömän siirtymisen viitattuun dokumenttiin. WWW-liikenteessä hyperteksti on muodostanut pohjan WWW-sivustojen rakenteelle ja verkostolle. Hyperlinkkien avulla on voitu tehdä viittauksia toisiin sivustoihin. Hyperlinkkien ansiosta on saatu aikaan toisiinsa kytkeytyvä rakenne, joka muodostaa loogisen kokonaisuuden itsenäisistä toisistaan riippumattomista osista. Hyperteksti on WWW-sivustoilla muotoiltu Hypertext Markup Language -merkintäkiellellä. Hypertext Markup Language, eli HTML, mahdollistaa tekstinmuotoilun ja -rakenteen määrittämisen. Uusimmilla HTML-versioilla on mahdollista myös upottaa erilaisia elementtejä, kuten video- tai audiotiedostoja, dokumenttiin. WWW-sivustot ovat kehittyneet alkuaikojen yksinkertaisista tekstisivustoista merkittävästi. Sivustota sisältävät nykypäivänä paljon muutakin sisältöä kuin

tekstiä. Sivustojen ominaisuuksia on lisätty mahdollistamalla HTML:n ulkopuolisten komentojen sisällyttäminen sivustojen lähdekoodiin. HTML-koodiin sisällytetyt ohjelmointi- tai kuvauskielet mahdollistavat esimerkiksi interaktiivisiatoiminnallisuuksia HTML-dokumenttiin. WWW-sivustoilla käytettyjä koodikieliä ovat esimerkiksi CSS-tyyliohjeet, Javascript-komentosarjakieli ja Flash-kehitysympäristö.

2.1 Protokollan toiminta

HTTP-protokolla perustuu HTTP-pyyntöihin. HTTP-pyynnöt yksinkertaistettuna sisältävät tiedon halutusta resurssista ja tiedon palvelimesta, jolta haluttu resurssi pyydetään. WWW-liikenteessä haluttu resurssi ilmaistaan Uniform Resource Locator:in avulla. URL pitää sisällään sekä tiedon palvelimesta että halutusta resurssista. Esimerkiksi "www.aalto.fi/fi/" on URL, jossa "www.aalto.fi" ilmaisee palvelimen osoitteen, eli domainin, ja "/fi/" ilmaisee halutun resurssin. Tässä tapauksessa "/fi" määrittää haluttavan resurssin palvelimen suomenkielisestä osiosta. Suomenkielisen osion sisältä haetaan vielä varsinaista resurssia, joka on tässä tapauksessa hakemistopolun juuri eli etusivu. HTTP-protokolla toimii TCP/IP-protokollapinossa, joten voidakseen todella saada IP-verkosta tuon halutun palvelimen, täytyy HTTP:n selvittää "www.aalto.fi"-palvelimen IP-osoite. IP-osoitteella yksilöidään verkkoa käyttävä laite, joka on tässä tapauksessa palvelin. Laitteen IP-osoitteen selvityksessä käytetään apuna Domain Name System-palvelua. Domain Name System, eli DNS, muuntaa domainnimiä IP-osoitteiksi ja päinvastoin. DNS-kyselyn avulla asiakas saa selvitettyä "www.aalto.fi"-nimen avulla palvelimen IP-osoitteen. Tämän jälkeen HTTP-protokollan kannalta kriittisimmät tiedot HTTP-pyynnön tekemiseksi on asiakkaan tiedossa. Tämän jälkeen asiakas on valmis lähettämään HTTP-pyynnön verkkoon prosessoitavaksi.

HTTP-protokolla on pyyntö-vastaus-protokolla. Protokollan mukaan asiakas pyytää sisältöä ja palvelin vastaa tähän pyyntöön. Alunperin HTTP-protokolla on suunniteltu toimimaan periaatteella, joka loi uuden TCP-yhteyden jokaista HTTP-pyyntö-vastaus kokonaisuutta varten. Kuitenkin protokollan versiosta 1.1 lähtien HTTP on tukenut myös sitä, että yhden TCP-yhteyden aikana voidaan käydä yksi tai useampia pyyntö-vastaus-neuvottelua. Neuvottelujen mahduttaminen yhden yhteyden sisään on lähtökohtaisesti tarkoitettu parantamaan verkon käytettävyyttä, kun asiakkaan ei ole tarvetta avata useita peräkkäisiä TCP-yhteyksiä palvelimelle. TCP-yhteyden avaaminen vaatii aina tietyn prosessin, jonka välttäminen poistaa viivettä ja parantaa siten asiakaskokemusta.

Nro.	**Source**	**Destination**	**Protocol**	**Info**
1	192.168.0.12	62.241.198.245	DNS	Standard query A www.aalto.fi
2	62.241.198.245	192.168.0.12	DNS	Standard query response A 192.230.77.222
3	TCP-yhteyden avaus, joka käsitellään luvussa 4.1			
4	192.168.0.12	192.230.77.222	HTTP	GET /fi/ HTTP/1.1
5	192.230.77.222	192.168.0.12	HTTP	HTTP/1.1 200 OK (text/html)

Taulukko 2.1: HTTP-pyyntö

Taulukossa 2.1 on esitettynä yhteyden avaamiseen liittyvät viestit silloin kun asiakas lähettää ensimmäisen HTTP-pyynnön palvelimelle. Tässä esimerkisssä asiakas, eli 192.168.0.12, tavoittelee www.aalto.fi-domainissa olevaa palvelinta. Yhteys alkaa DNS-kyselyllä (1), jonka asiakas lähettää DNS-palvelimelle, 62.241.198.245. DNS-palvelin vastaa kyselyyn (2), ja palauttaa asiakkaan tietoon www.aalto.fi-domainiin kuuluvan IP-osoitteen tai IP-osoitteet, kuten tässä tapauksessa 192.230.77.222-osoitteen. Tämän jälkeen asiakas avaa TCP-yhteyden palvelimelle (3), TCP-yhteyden avaus käydään tarkemmin läpi luvussa 4.1. Kun TCP-yhteys on saatu avattua, voi asiakas lähettää varsinaisen HTTP-pyynnön (4). Tässä tapauksessa HTTP-pyyntö on "GET /fi/", eli asiakas pyytää palvelimen suomenkielisen osion juurta joka yleensä sisältää sivuston etusivun. Palvelin vastaa lopulta HTTP-pyyntöön 200 OK -vastauksella (5), joka käytännössä tarkoittaa, että pyyntö on hyväksytty ja täytetty. Vastauksen liitteenä palvelin lähettää sivuston sisällön. Tässä vaiheessa sivustosta on saatu asiakkaan haltuun vasta ensimmäinen osa laajahkoa www.aalto.fi:n etusivua. Tätä ensimmäistä pyyntö-vastaus-ketjua seuraa useita GET-pyyntöjä palvelimelle, joiden avulla ladataan sivuston muuta sisältöä. Muulla sisällöllä tarkoitetaan esimerkiksi kuvia ja mainoksia, muotoilua ja mahdollisia toiminnallisialisäosia, joita sivustoon on sisällytetty. Näiden pyyntöjen lisäksi on melkein yhtä pitkä lista samaan sivustopyyntöön liittyviä GET-pyyntöjä, jotka on osoitettu jollekin muulle domainille kuin www.aalto.fi:lle. Ulkopuolisille palvelimille liittyviin GET-pyyntöihin liittyy lähes poikkeuksetta uusia DNS-kyselyjä. Lähtökohtaisesti palvelimet ovat asiakkaalle tuntemattomia, minkä takia URL täytyy aina selvittää DNS-kyselyllä. DNS-kysely voidaan välttää sillä, että asiakas on tallentaa muistiinsa usein käytettyjen sivustojen osoitteita. Kuitenkin edelleen

jos sivusto on tuntematon asiakkaan täytyy suorittaa DNS-kysely. Lisäksi vähintään jokainen eri palvelimeen kohdistuva GET-pyyntö vaatii luonnollisesti myös TCP-yhteyden luonnin kyseiseen palvelimeen. Kaiken kaikkiaan www.aalto.fi-sivuston etusivun lataaminen tuotti noin kolmetuhatta HTTP-viestiä palvelimien ja asiakkaan välille, mukaan lukien kaikki pyynnöt ja vastaukset, joita yhteyden aikana tuli.

2.1.1 HTTP otsake

HTTP-protokollan otsakkeet ovat olleet alusta alkaen ihmisen luettavissa. Otsakkeet sisältävät ASCII-merkistön merkkejä, ja protokollan tietokentät ovat pääosin englanninkielisiä ja itsensäselittäviä. Merkkijonojen käyttäminen protokollan otsakkeessa on tehnyt protokollan käytöstä helppoa ja mahdollista jo varhaisilla päätelaitteilla, joissa ei välttämättä ole ollut valmista WWW-selainta. Alunperin viestinvaihto on tapahtunut käyttäjältä suoraan palvelimelle pääteyhteyksien ylitse, jolloin protokollan ymmärrettävyys on ollu merkittävä ominaisuus. Vikatilanteiden tutkiminen HTTP-viestivirrasta on ollut helpompaa, sillä ihmisen on ollut verrattain helppo ymmärtää sisältö HTTP-otsakkeissa ja vikailmoituksissa.

```
GET /fi/ HTTP/1.1\r\n
Host: www.aalto.fi\r\n
Connection: keep-alive\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
Upgrade-Insecure-Requests: 1\r\n
User-Agent: Mozilla/5.0 (X11; Linux i686 (x86_64)) AppleWebKit/537.36 (KHTML, like Gecko) Chro
me/45.0.2454.101 Safari/537.36\r\n
Accept-Encoding: gzip, deflate, sdch\r\n
Accept-Language: fi-FI,fi;q=0.8,en-US;q=0.6,en;q=0.4\r\n
\r\n
```

Kuva 2.1: HTTP 1.1 GET-pyyntö

Kuvassa 2.1 näkyy HTTP-protokollan version 1.1 mukainen GET-pyyntö. Oheisella pyynnöllä käyttäjä pyytää palvelimelta www.aalto.fi sivuston sisältöä. Kuten kuvasta nähdään, on pyynnön kaikki otsakekentät ihmisluettavassa muodossa. Kuten luvussa 2.1 todettiin, riittäisi HTTP-pyyntöön merkittävästi vähemmänkin tietoa. Kuitenkin oheinen pyyntö on nykyaikaisen selaimen tuottama, ja se sisältää olennaisia tietoja. Esitettyjen tietojen avulla palvelin voi pyrkiä huomioimaan mahdollisimman tarkasti asiakkaan tarpeet. Oheistietojen joukossa ilmaistaan esimerkiksi asiakkaan verkkoselain, jonka avulla palvelin voi päätellä käyttääkö asiakas mobiililaitetta vai tietokonetta, onko asiakkaalla käytössä uudenaikainen selain vai kenties joku vanha versio ja niin edelleen. Näiden tietojen avulla palvelin voi lähettää asiakkaalle opti-

moidun sivuston, jonka pitäisi aueta asiakkaan päätelaitteella mahdollisimman toimivasti. Tämänkaltainen toiminta parantaa asiakaskokemusta merkittävästi, sillä sivuston aukeaa toivotunlaisena asiakkaalla päätelaitteesta riippumatta. Lisäksi palvelin voi halutessaan tarjota erilaista sisältöä asiakkaan selaimesta tai päätelaitteesta riippuen.

Ihmisluettavan protokollan osittainen ongelma verkkoliikenteen kannalta on sen vaatima tiedonmäärä. ASCII-merkistön mukaiset merkit vaativat yleensä esitysmuotoonsa kahdeksan bittiä, eli yhden tavun, jokaista esitettyä merkkiä kohden. Tässä tapauksessa yllä esitetty GET-pyyntö sisältää yhteensä 418 merkkiä eli 418 tavua. Verrattuna täysin binääriseen esitysmuotoon 418 tavua on valtava määrä tietoa. 418 tavua sisältää 3344 bittiä. Teoriassa tällä määrällä tietoa voisimme ilmaista 3344 asiaa jotka olisivat joka päällä tai pois päältä. Käytännössä kuitenkin binäärisetkin protokollat käyttävät usein bittien yhdistelmiä asioiden ja tietokenttien sisällön ilmaisuun, jolloin sisältö voi olla monimuotoisempaa ja muuttuvampaa. Yhtenä suurena erona siis binääriseen protokollan ja ihmisluettavan protokollan välillä on niiden tuottaman tiedon määrä. Kuvassa 2.1 esitetyn GET-pyynnön voisi siis parhaassa tapauksesssa esittää vain muutamalla tavulla per kenttä, jos se olisi toteuttu binäärimuodossa. Binääriesitys olisi merkittävästi tiivimmin pakattua tietoa kuin ihmisluettavan protokollan tarjoama vaihtoehto. Binääriesitys ei käytännössä muuttaisi tai vaikuttaisi millään tavalla protokollan kantamaan tietoon itsessään, ainoana erona on tiedon esitysmuoto. Googlen mukaan suurimmillaan nykyisten WWW-sivujen koko saattaa nousta jopa yli kahden kilotavun. Googlen mukaan tyypillinen WWW-sivun koko olisi noin 750 tavua, mukaan lukien kaikki evästeet ja muut lisäosat jotka kuuluvat olennaisena osana nykyisiin sivustoihin[13].

Binäärisyys ei kuitenkaan sinällään ratkaise mitään, sillä merkittävää on myös protokollan otsakkeen suunnittelu kokonaisuutena. Binäärinenkin protokolla voi olla tehoton, josseivät sen sisältämät kentät sisällä sellaista tietoa, joka olisi välittömästi käytettävissä ja sellaisenaan hyödynnettävissä. Esimerkiksi HTTP:n GET-pyyntö pitää sisällään melko tehokkaasti vain ja ainoastaan sen tiedon, joka on merkittävää protokollapinon toiminnan kannalta HTTP-pyynnön toteuttamiseksi. Kuitenkin GET-pyyntö ei pidä sisällään mitään sellaista tietoa, minkä takia sitä ei voisi esittää myöskin binäärisessä muodossa. GET-pyynnön ihmisluettavuus on säilynyt sellaisenaan, sillä palvelinten prosessointikyky ja verkon kuljetuskapasiteetti ovat toistaiseksi riittäneet verrattain hyvin pyyntöjen käsittelyyn ja välittämiseen. Toinen merkittävä syy ihmisluettavuuden säilymiseen on ollut se ettei HTTP-protokolla ole ollut helposti korvattavissa, eikä sen muuttaminen binääriseksi ole ollut täysin triviaali prosessi. Ylipäänsä nykyisten verkkoprotokollien muutoksen

hidasteena on ollut niiden saavumattama valtaisa suosio. Muutoksen tekeminen vaikuttaisi välittömästi valtavaan määrään erilaisia laitteita.

2.2 HTTP/1.1 version ongelmat

HTTP/1.1 versiossa on useita ominaisuuksia, jotka eivät täysin sovellu nykyiseen verkkoliikenteeseen. Ongelmien synty on ymmärrettävää, sillä protokollan määrittely on tapahtunut jo vuonna 1997. Protokollan julkaisun jälkeen verkkoliikenne on muuttunut merkittävästi. Osa protokollan ongelmista ovat rakenteellisia, minkä takia niiden korjaaminen erilaisilla lisäosilla tai -ominaisuuksilla ei ole ollut mahdollista. IETF:n Hypertext Transfer Protocol -työryhmä, jonka tavoitteena oli uudistaa HTTP-protokollaa, listasi työskentelynsä alussa merkittävimpiä ongelmia. Työryhmän mukaan ongelmat muodostuvat HTTP1.1 version määrittelyssä tapahtuneista suunnitteluvirheistä tai epätarkasti dokumentoiduista toiminnallisuuksista. Työryhmän tavoitteena oli selkeyttää protokollan ominaisuuksien määrittelyjä ja poistaa epäjohdonmukaisuuksia määrittelyistä. Lisäksi protokollan määrittelyssä on esitetty poistettavaksi ominaisuuksia, joita todellisuudessa käyttäjät eivät ole laajalti ottaneet käyttöön. [20]

HTTP-protokolla on ollut ongelmallinen rakenteensa vuoksi verkon kasvaessa. Protokollan mukaan yksi HTTP-pyyntö voi pitää sisällään vain yhden HTTP:n toimintametodin. HTTP:n toimintametodeja on esimerkiksi GET, POST, CONNECT ja PUT. Näistä yleisin on GET-pyyntö, jolla asiakas pyytää palvelimelta haluttua resurssia. Muut metodit suorittavat esimerkiksi uudelleenohjauksia palvelimien tai välityspalvelimien välillä ja tiedostojen siirtoa asiakkaalta palvelimelle. Ongelma muodostuu protokollan tehottomuudesta, sillä kun vain yksi toiminto voidaa suorittaa pyyntöä kohden kasvaa protokollan toteutukseen liittyvän liikenteen määrä suhteessa todelliseen tietoon hyvinkin suureksi. Lisäksi siirrettävä tieto liikkuu HTTP-yhteydessä hyvin hitaasti verrattuna verkon kapasiteettiin. HTTP-yhteys on alunperin tarkoitettu ja suunniteltu vuoropuheluksi. Nykyiset WWW-sivustot kuitenkin koostuvat useista käytännössä itsenäisistä osista, joiden välittäminen vuoropuheluna on hidasta ja tehotonta. Tätä rajoitusta on kierretty HTTP-pipelining-tekniikalla, joka mahdollistaa usean HTTP-pyynnön lähettämisen yhden yhteyden sisällä kerrallaan. HTTP-pipelining mahdollistaa sen, että asiakkaan ei tarvitsee odottaa palvelimen vastausta jokaiseen lähetettyyn HTTP-pyyntöön, ennen kuin se voi lähettää uuden pyynnön. Toinen pitkään käytössä ollut tapa kiertää rajoitusta on avata useita rinnakkaisia TCP-yhteyksiä, jolloin jokaisessa yhteydessä voidaan lähettää erilli-

nen HTTP-pyyntö. Usean rinnakkaisen TCP-yhteyden avaaminen ei kuitenkaan ole verkon kokonaissuorituskyvyn kannalta hyvä ratkaisu, syyt tähän käsitellään tarkemmin luvussa 4.3. [9, 10]

HTTP-protokolla vaatii verrattain paljon protokollakohtaisia tietoja suhteessa siirrettävän tiedon määrään. Esimerkiksi otsakkeessa on paljon tietokenttiä, jotka toistuvat jokaisessa lähetetyssä HTTP-viestissä. Kuvassa 2.1 nähdään HTTP-pyynnön rakenne. Kuvassa esiintyvä viesti toistuu jokaisen sivuston osan kohdalla uudestaan, kun asiakas pyytää palvelimelta uutta osaa sivustosta. Asiakkaan pysyessä koko ajan samalla sivustolla, ainut asia mikä pyynnössä muuttuu on ensimmäinen rivi ja siitäkin vain resurssi, johon pyyntö kohdistetaan. Kuvassa olevassa pyynnössä pyyntö on kohdistettu /fi/ resurssiin, mutta se voisi yhtä hyvin olla jokin sivuston alisivuista. Kaikki muut tiedot kyseisessä pyynnössä pysyvät lähtökohtaisesti samana koko yhteyden ajan. HTTP-protokollan mukaan muuttumattomatkin tiedot lähetetään joka kerta, jokaisessa pyynnössä uudelleen asiakkaalta palvelimelle.Tämä tuottaa valtavan määrään ylimääräistä tietoa, joka periaatteessa on tiedossa molemmilla osapuolilla yhteyden avaamisen jälkeen ensimmäisen viestin saavuttua. Tämä yhdistettynä siihen, että HTTP-protokolla lähettää ASCII-merkistöllä koodattuja selkokielisiä viestejä, tuottaa protokolla kokonaisuutena valtavan määrä tietoa siirrettäväksi ja käsiteltäväksi.

Luku 3

Hypertext Transfer Protocol/2.0

HTTP/2 on HTTP-protokollan viimeisin versio, edellinen yhtä merkittävä päivitys HTTP-protokollaan on julkaistu vuonna 1997 kun julkaistiin HTTP 1.1[11]. HTTP/2 päivityksen on kehittänyt IETF:n Hypertext Transfer Protocol -työryhmä. HTTP/2.0 protokolla on tarkoitettu sovelluskerroksen protokollaksi, tarjoamaan sovellusrajapinnan Hypertext-dokumenttien välittämiselle. Tässä luvussa käsitellään HTTP/2.0 protokollan lyhyt kehityshistoria, toimintaperiaatteet ja rakenne.

3.1 HTTP/2-protokollan tavoitteet

Vuonna 2012 HTTP-protokollan uudistamisen parissa työskenteli useat työryhmät. Virallisen standardin julkistaisi IETF, ja sen kehittämisestä vastaisi HTTP-työryhmä. HTTP-protokollan kehittämistä aktiivisesti työsti ainakin Google, Microsoft ja Varnish Cache Project[3, 23, 25]. Kevään 2012 aikana kaikki kolme työryhmää julkaisi "Internet-Draftin" kuvaamaan HTTP-protokollaan tarvittuja uudistuksia. Internet Draftilla tarkoitetaan julkista esittelyä verkkoon tarkoitetusta protokollasta. Esittelyn tarkoituksena on kerätä kommenntteja muilta kehittäjiltä ja käyttäjiltä. Googlen esitys korvaajaksi oli SPDY-protokolla, Microsoft tarjosi HTTP Speed+Mobility-protokollaa ja Varnish Cache Project antoi oman esityksensä verkko-ystävällisestä HTTP-päivityksestä. Projektit lähtivät erilaisista lähtökohdista. Google tarjosi kokonaan uuden protokollan, joka säilyttäisiä ainoastaan alkuperäiset HTTP-pyynnöt. Microsoft pyrki hyvinkin samanlaiseen toteutukseen kuin

Google. Varnish Cache Project pyrki merkittävästi pienemmillä muutoksilla ja päivityksillä tuottamaan vastaavanlaisia hyötyjä kuin Microsoft ja Google. IETF:n HTTP-työryhmän ensimmäinen nk. "Initial-draft" julkaistiin käytännössä suorana kopiona SPDY:stä. Initial-draftilla tarkoitetaan IETF:n esittelemään protokolla kuvausta verkkoon tarkoitetusta protokollasta. Initial-draftin jälkeen IETF keräsi kommentteja protokollasta. SPDY:hyn pohjautunut Initial-draft sai hyvän vastaanoton ja myöhemmin sen pohjalta julkaistiin varsinainen HTTP/2 -protokolla[3].

IETF:n HTTP -työryhmän erityisinä tavoitteina ja kehityksen kohteina uutta protokollaa luodessa oli mahdollistaa HTTP:n kommunikointimekanismin laajat soveltamismahdollisuudet, otsakkeiden pakkauksen parantaminen ja palvelimelle push-ominaisuuden mahdollistaminen. Uuteen HTTP-protokollaan oli tarkoitus toteuttaa neuvottelumekanismi, joka mahdollistaa yhteysneuvottelun, sekä nykyistä protokollaa edeltävien, että tulevien HTTP-protokollaversioiden kanssa. Lisäksi pyrkimyksenä oli mahdollistaa minkätahansa muun HTTP URL:ja käyttävän protokollan kanssa käytävä yhteysneuvottelu. Otsakkeiden pakkaus koettiin tarpeelliseksi uudistaa. Nykyinen HTTP-otsake oli todettu erittäin raskaaksi tavaksi toteuttaa vastaava toiminnallisuus nykyisessä WWW-liikenteessä, kuten todettiin luvussa 2.1.1. Työryhmä esitti vaihtoehtona, että pakkauksen mahdollistamisen ohella myös mahdollistettaisiin otsakkeiden osiointi.

Palvelimen push-toiminnallisuudella tarkoitetaan ominaisuutta, jolla palvelin voisi lähettää asiakkaalle sisältöä ilman, että asiakas sitä erikseen pyytää. Tällä pyritään erityisesti nopeuttamaan tilanteita, joissa palvelin käytännössä tietää jo ennalta mitä asiakas tulee seuraavaksi pyytämään. Tälläinen tilanne syntyy nykyisissä WWW-sivuissa usein, sillä sivustot koostuvat monista elementeistä. Useista elementeistä koostuvien sivustojen jokainen elementti on tähän asti vaatinut asikkaalta aina erillisen pyynnön. Uusi push-ominaisuus mahdollistaisi palvelimen "työntää"asiakkaalle sivuston osat, joita asiakas tulisi tarvitsemaan, joka tapauksessa ladatakseen koko sivuston. [20]

Google julkaisi ensimmäisen versionsa SPDY-protokollasta vuonna 2009. SPDY:n kehittäjien tarkoituksena oli luoda ikkunoiva kerros TCP:n ja HTTP:n väliin, jonka avulla HTTP-liikenne saataisiin sopimaan paremmin nykyisen WWW-liikenteen ehdollisuuksiin. Tavoitteena oli, että HTTP- ja TCP-protokollien toiminnallisuudet säilyisivät muuttumattomina. Lisäksi pyrittiin siihen, ettei HTTP:tä käyttävien sovellusten toimintamekanismejakaan olisi tarpeen muuttaa SPDY:n takia. SPDY:n kehittäjien tavoitteena oli parantaa kolmea perustavanlaatuista ominaisuutta silloisesssa WWW-liikenteessä.

Yhtenä tavoitteena oli HTTP pyyntöjen limittäminen yhdelle SPDY-yhteydelle, ilman minkäänlaista rajoitusta HTTP-pyyntöjen määrästä. Tämän ominaisuuden tavoitteena oli helpottaa TCP-protokollan toimintaa. TCP-protokollan toiminta helpottuisi, sillä SPDY avaisi vain yhden TCP-yhteyden ja limittäisi HTTP-pyynnöt sen sisään. Kaikki HTTP-pyynnöt yhdelle asiakkaalle kulkisivat yhdessä SPDY-yhteydessä usean rinnakkaisen TCP-yhteyden sijaan. SPDY luokittelee pyynnöt tärkeisiin ja tavallisiin pyyntöihin. Tämäkin ominaisuus rakentuu sen ajatuksen ympärille, että verkkosivujen tärkeät osat saadaan ladattua asiakkaalle mahdollisimman nopeasti. Kuten luvussa 2 todetaan nykyiset verkkosivut sisältävät paljon myös osioita, jotka eivät ole asiakkaalle varsinaisesti välttämättömiä sivun esittämisen kannalta. Pyyntöjen priorisoinnilla voidaan parantaa sivun tärkeiden osien latautumisen nopeutta asiakkaalle. Kun asiakkaalle on jo saatu tärkeimmät osiot sivustoa näkyviin voidaan vähemmän tärkeät osat sivustoa ladata taustalla samaan aikaan kun asiakas jo tarkkailee esitettävissä olevaa sivustoa.

Kolmas merkittävä tavoite SPDY:llä on pienentää ja tiivistää otsakkeiden kokoa. Perusteluina kehitysryhmä esittää sen että HTTP-otsakkeet sisältävät monilta osin samaa tietoa rinnakkaisissa pyynnöissään. Rinnakkaisissa pyynnöissä muuttamattomana oleva tieto voidaan poistaa, ja sitä voidaan päivittää ainoastaan silloin kun se muuttuu. Otsakkeiden aiheuttaman lisäkuorman vaikutus verkonsuorituskyvylle on merkittävä ja siksi otsakkeiden tiivistäminen koettiin erityisen tärkeäksi[2].

Lisäksi jo ensimmäinen versio SPDY-protokollasta esittelee palvelimen push-toiminnon. Tosin SPDY:n ensimmäisen version kuvauksessa todetaan, ettei vielä siinä vaiheessa esitelty toiminto ole lähelläkään toimintavalmista. Protokollan kuvauksessa todetaan myös toteuttamisen vaativan vielä merkittävää työtä ja paneutumista aiheen ympärillä. Kuitenkin jo tässä vaiheessa ajatus push-toiminnosta oli olemassa, ja SPDY:n protokolla kuvaus antaa jo aikaisessa vaiheessa hyvän viitekehyksen sille kuinka se voitaisiin toteuttaa. Luvussa 3.2 käsitellään enemmän kuinka palvelimen push-toiminto todellisuudessa toteutettiin lopulliseen HTTP/2-protokollaan. [2]

3.2 HTTP/2 toiminnallisuudet

HTTP/2 esittelee useammankin uuden toiminnallisuuden HTTP-protokollalle. Näistä merkittävimpinä ovat palvelimen push-toiminto ja viestivirtojen limitys (eng. multiplexing). Molemmat toiminnot ovat olleet pitkään kaivattuja HTTP-protokollaan. HTTP1.1-protokollaan esitelty HTTP-pipelining

toiminnallisuus pyrki toteuttamaan limityksen jo aiemmassa HTTP-protokollan versiossa. HTTP-pipelining ei kuitenkaan todellisuudessa toteuta täysin samaa kuin mitä HTTP/2-protokollan limitys, mutta ajatusmalli taustalla on kuitenkin vastaava. HTTP-pipelining pyrki toteuttamaan limityksen erillisten pyyntöjen peräkkäisellä lähettämisellä ennen kuin edeltäviin pyyntöihin on saatu vastausta. HTTP/2 protokollassa limitys on toteuttu limittämällä pyynnöt aidosti yhden yhteyden sisään kulkemaan.[9]

Push-toiminnallisuus mahdollistaa palvelimen lähettää asiakkaalle sen tarvitsemia elementtejä sivustosta jo ennen kuin asiakas esittää erillistä pyyntöä elementtejä saadakseen. Käytännössä tällä toiminnallisuudessa mahdollistetaan WWW-sivustojen sisältämien erillisten sivuston osien, kuten kuvien ja linkkien, lähettäminen käyttäjälle ilman erillistä pyyntöä. Push-toiminto perustuu siihen, että palvelin käytännössä tietää asiakkaan seuraavan pyynnön sivuston rakenteen ansiosta jo ennen kuin asiakas on varsinaista pyyntöä esittänyt. Palvelimen käyttäessä push-toimintoa se todellisuudessa lähettää käyttäjälle sekä käyttäjän tarvitseman sivuston osan, että käyttäjän esittämän kuvitteellisen pyynnön. Kuvitteellinen pyyntö on pyyntö, jonka asiakas esittäisi tarvitessaan kyseistä elementtiä, mutta jota asiakas ei tässä tapauksessa ole vielä lähettänyt palvelimelle. Tämä kuvitteellinen pyyntö lähetetään käyttämällä Push_promise-kehystä.

Toinen HTTP/2-protokollan esittelemä merkittävä toiminnallinen uudistus on viestivirtojen limitys. Limitys mahdollistaa asiakkaan ja palvelimen välille usean täysin toisistaan erillisen viestinvaihdon, jotka kuitenkin tapahtuvat saman yhteyden sisällä. Käytännössä limitys toteutetaan viestivirtojen (eng. stream) avulla. Jokainen pyyntö-vastaus-ketju on yhdistetty uniikkiin virtaan. Käyttäjän luomien virtojen virtatunnukset ovat parittomia lukuja ja palvelimen luomat virtatunnukset ovat parillisia. Virta luodaan ensimmäisen kerran asiakkaan Headers-kehyksellä tai palvelimen Push_promise-kehyksellä, käytännössä siis valtaosa virroista luodaan asiakkaan puolella. Jokaisessa HTTP/2 kehyksessä on virtatunnus, jonka avulla jokainen viesti on käsittelijän toimesta tunnistettavissa joko palvelimen tai asiakkaan luomaksi. Oletusarvoisesti virtojen määrälle ei ole esitetty minkäänlaista rajoitusta, mutta käytännössä palvelimelle on luotu mahdollisuus Settings-kehyksen avulla rajoittaa asiakkaalle myönnettyjen resurssien määrää ja siten myös virtojen määrää. [24]

Merkittävä toiminnallisuus HTTP/2 protokollassa on SPDY-protokollaan kehitetty otsakkeiden pakkaus. Otsakkeiden pakkauksen todettiin SPDY:n käytössä merkittävästi pienentävän otsakkeiden kokoa. Otsakkeiden pakkauksessa keskitytään erityisesti niiden otsakekenttien pakkaamiseen jotka

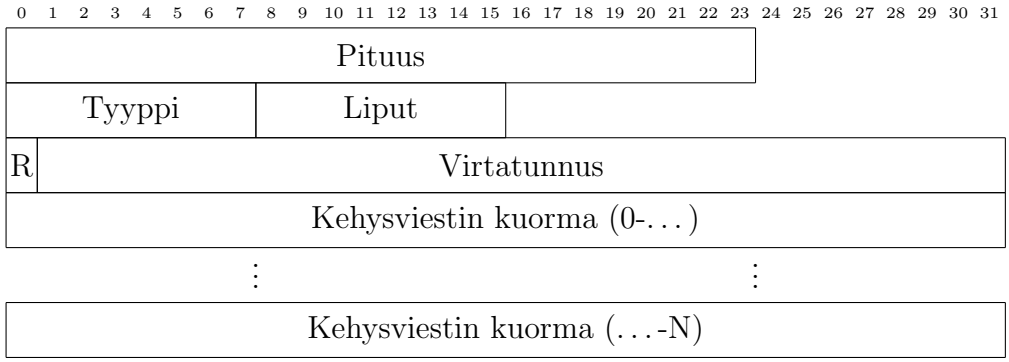
toistuvat muuttumattomina HTTP-viestistä toiseen. SPDY-protokollassa käytetty DEFLATE-pakkaus todettiin melko nopeasti tietoturvariskiksi, jonka myötä sen käytöstä luovuttiin[21]. HTTP/2 -protokollaan kehitettiin vastaava pakkausmenetelmä, HPACK. Lisäksi HPACK-menetelmää kehitettiin entisestään ja sillä on saavutettu erittäin merkittäviä eroja otsakkeiden koossa [17]. HPACK on tarkoitettu yksinkertaiseksi ja helpoksi tavaksi vähentää HTTP-liikenteessä esiintyvää toistoa otsakkeissa. Yksinkertaisen toimintamallin tarkoituksena on helpottaa käytännön toteutuksia ja siten vähentää riskiä tietoturvaongelmille. HPACK-spesifikaatio listaa olemassa olevat, tiedostetut tietoturvariskit, ja esittelee niihin liittyvät tarvittavat toimenpiteet. Käytännössä tietoturvariskit HPACK-pakkauksen kanssa ovat aina seurauksia tarkoituksellisesta toiminnasta, kuten palvelunestohyökkäyksestä, jolla on tarkoitus saada aikaa poikkeustilanne palvelimella. Lähinnä ongelmat HPACK:in tapauksessa liittyvät muistin hallintaan palvelimella, sillä erityisesti pakkauksen purkaminen vaatii muistiresursseja. Käytännössä kuitenkin HPACK:in tietoturvariskit ovat pieniä, eikä niiden olemassa olo estä HPACK:in käyttöä. Tarkoituksellisia hyökkäyksiä vastaan täytyy palvelimen ylläpitäjän varautua nykyisessä verkkoliikenteessä kuitenkin, eikä HPACK aiheuta tässä tapauksessa poikkeusta. [21]

3.3 HTTP/2 protokollan rakenne

HTTP/2 tukee kaikkia samoja metodeita, otsakkeita ja serveritiloja joita HTTP/1.1-protokolla tuki. Käytännössä HTTP-protokolla ei uudistu toiminnassaan, vaan enemmänkin tavassa, jolla se toimii. Uuden standardin on tarkoitus määrittää HTTP-protokolla niin, että alla olevalle siirtoprotokollalle jää aidosti tehtäväkseen ainoastaan vastata palvelimen ja asiakkaan välillä tapahtuvasta tiedonsiirrosta. Aiemmin HTTP-protokolla on hankaloittanut siirtoprotokollan tehtävää esimerkiksi luomalla useita rinnakkaisia HTTP-sessioita yhden asiakkaan ja palvelimen välille. Useiden yhteyksien luominen rinnakkain on hankaloittanut siirtoprotokollan jononhallintaa, sillä esimerkiksi TCP-protokollan sisäänrakennettu ruuhkanhallinta tapahtuu yhteyskohtaisesti. Yhteyskohtaisesti tapahtuvan ruuhkanhallinnan tapauksessa rinnakkaiset yhteydet samalla asiakkaalla muodostavat kilpailutilanteen asiakkaan omien yhteyksien välille, mikä ei ole omiaan nopeuttamaan kokonaisuuden palvelunlaatua. [8]

3.3.1 Otsake

HTTP/2 protokolla käyttää käytännössä samoja otsakkeita kuin HTTP-protokolla aiemminkin. Erona kuitenkin aiempaan on se, että HTTP/2-protokolla on täysin binäärinen. Otsakkeet on siis muutettu binäärisiksi, mikä luonnollisesti tiedon pysyessä samana pienentää lähetettävän tiedon määrää. Toinen suuri muutos otsakkeiden binäärisiksi muuttamisen ohella on kehysviestit (eng. Frame). HTTP/2-sessio koostuu käytännössä kolmesta vaiheesta, yhteyden muodostamisesta, yhteydestä ja yhteyden purkamisesta. Yhteyden muodostamisen jälkeen HTTP/2-sessiossa voi alkaa kehysviestien vaihto. Kehysviesti alkaa aina samalla 9-oktettia pitkällä kehysotsakkeella, jota seuraa sen jälkeen kehysviestistä riippuen vaihtelevan pituinen hyötykuorma. Kehysviestien yhteinen otsake sisältää viestin pituuden, tyypit, liput ja virtatunnuksen. Tässä luvussa käsitellään myöhemmin tarkemmin kehysviestien rakenne.



Kuva 3.1: HTTP/2 kehysviesti

Kehysviestien tarkoituksena on muodostaa selkeä kokonaisuus erilaisista toiminnoista protokollan sisällä. Yksittäinen viesti palvelimen ja asiakkaan välillä voi sisältää useita kehysviestejä, joista jokainen toimittaa omaa tehtäväänsä.

3.3.2 Kehysviestit

Kuten edellä on todettu kehysviestien tarkoituksena on selkeyttää HTTP/2 -protokollan rakennetta. Jokainen HTTP/2 -protokollan lähettämä viesti sisältää vähintään yhden kehyksen. Erilaisia kehysviestejä on yhteensä kymmenen kappaletta.

Data
Kehys kuljettaa viestivirran vaatimat hyötykuormat, käytännössä esimerkiksi HTTP-pyynnöt tai pyyntöjen vastaukset.

Headers
Kehys sisältää listan HTTP-protokollan otsaikkeista kyseisessä viestissä, sellaisena kuin ne HTTP1.1 protokollasta tunnetaan.

Priority
Kehyksellä lähettäjä määrittää viestivirran tärkeyden lähettäjän näkökulmasta.

Rst_stream
Kehyksellä on tarkoitus ilmaista toiselle osapuolelle viestivirran välittömästä katkaisusta.

Settings
Kehyksellä lähettäjä määrittelee halutun yhteyden aikaisen kommunikointitavan. Esimerkiksi asiakas voi ilmaista palvelimelle haluaako se palvelimen käyttävän push-toimintoa. Myös palvelin voi määrittää kehyksen avulla yhteyden viestintätapoja.

Push_promise
Kehyksellä palvelin tiedottaa asiakkaalle ennakkoon tulevasta palvelin push-toiminnosta.

Ping
Kehyksen avulla voidaan varmistaa, että yhteys on vielä olemassa, sekä yhteyden RTT-aika.

Goaway
Kehyksen tarkoituksena on ilmaista toiselle osapuolelle ettei uusia viestivirtoja tulisi avata nykyiseen yhteyteen. Goaway:n aikana palvelin kuitenkin huolehtii nykyiset viestivirrat loppuun. Kehys on tarkoitettu käytettäväksi esimerkiksi palvelimen huoltojen yhteydessä.

Window_update
Kehyksellä on tarkoitus mahdollistaa virran hallinta(eng. flow control), jonka avulla osapuolet voivat säädellä virtaa paremmin prosessointiresurssiensa suhteen.

Continuation
Kehys seuraa aina joko Headers tai Push_promise -kehyksiä, ja sen

tarkoituksena on kuljettaa mahdolliset ylijäävät osuudet varsinaisista HTTP-otsakkeista, jotka eivät mahdu varsinaiseen kehysviestiin.

Edellä esitellyillä kehysviesteillä on tarkoitus yksinkertaistaa, selkeyttää ja tehostaa HTTP-protokollan toimintaa. Kehykset tarjoavat kattavasti kaikki HTTP-yhteyden aikana tarvittavista toiminnoista. Lisäksi kehykset erottavat luontaisesti yhteyden ylläpitoon liittyvät toiminnot varsinaisista sivuston sisältöön liittyvistä toiminnoista. Esimerkiksi varsinaiset HTTP-otsakkeet on luotu vain osaksi yhtä kehystä, toisin kuin HTTP/1.1 protokollassa, jossa kaikki tieto palvelimen ja asiakkaan välillä oli sisällytetty varsinaiseen viestin otsakkeeseen. [24]

Luku 4

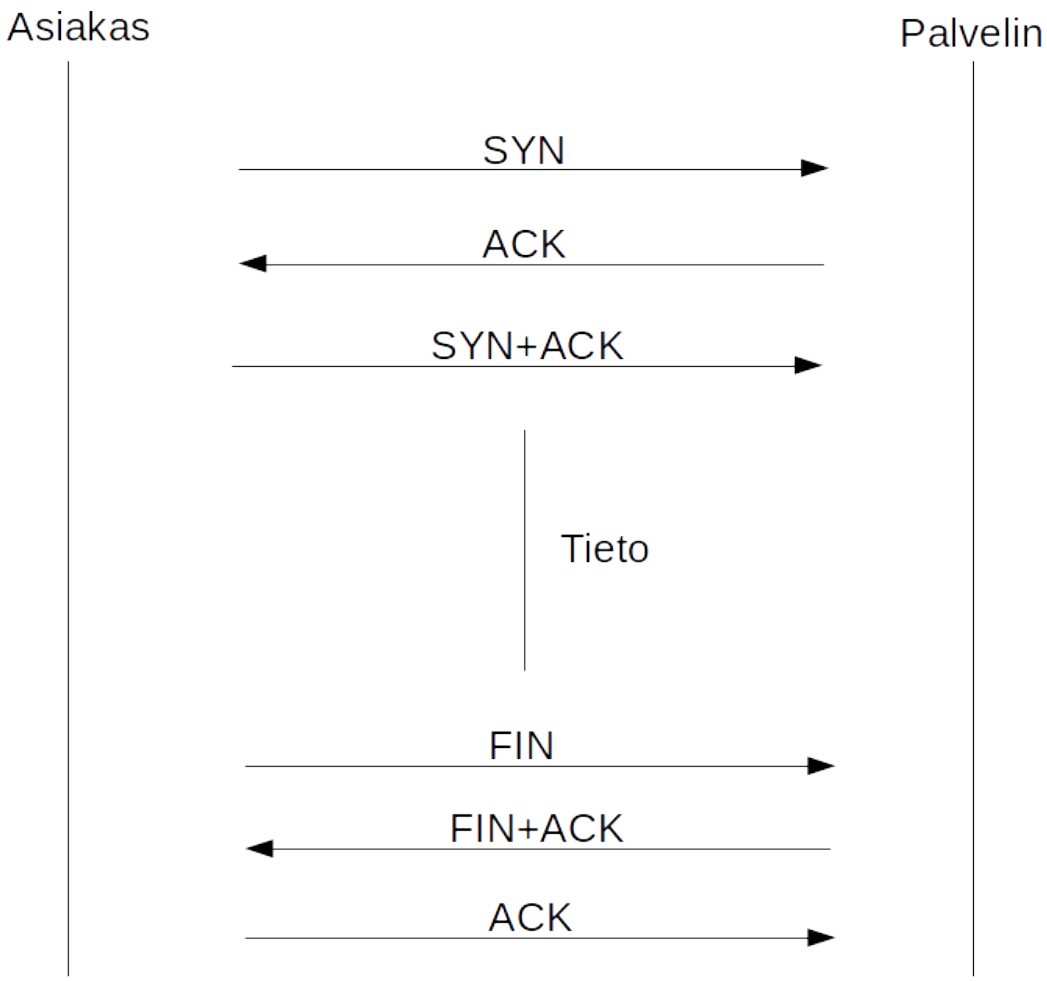
Transmission Control Protocol

TCP-protokolla on yksi internetin vanhimmista protokollista. Protokolla on pysynyt käytännössä muuttumattomana, mutta sen ominaisuuksia on monipuolistettu lisäosilla ja laajennuksilla. TCP:n tarkoitus on toimia kuljetuskerroksen protokollana WWW-liikenteessä ja vastata viestien kuljetuksesta IP-osoitteilla varustettujen päätelaitteiden välillä. TCP:n tarkoituksena on olla luotettava protokolla, joka kuljettaa lähetettävän paketin perille. Kuljetuskerroksen protokollana TCP huolehtii asiakkaiden puolesta kaikesta kuljetuskerroksella tapahtuvasta toiminnasta. Luotettavalla protokollalla TCP:n tapauksessa tarkoitetaan sitä, että TCP varmistaa pakettien kulun kuljetuskerroksella. Käytännössä TCP lähettää uudelleen paketteja, joita se huomaa hukkuneen kuljetuksen aikana. Luotettava protokolla myös varmistaa että kaikki lähetetyt paketit saapuvat perille oikeassa järjestyksessä niin, että niistä tiedon lukeminen onnistuu helposti. TCP protokollan toiminta on WWW-sivustojen käyttäjälle käytännössä näkymätöntä, eikä sen käyttäminen edellytä asiakkaalta yleensä minkäänlaisia toimenpiteitä.

4.1 TCP:n toimintaperiaatteet

TCP-protokolla aloittaessaan TCP-yhteyttä käyttää niin sanottua kolmitiekättelyä, jonka tarkoituksena on tiedottaa molempia päätelaitteita yhteyden laadusta ja tyypistä, sekä sopia varsinaisen yhteyden alkamisesta.

Kuvassa 4.1 on kuvattuna kolmitiekättely. Kolmitiekättely varmistaa TCP-yhteyden osapuolten valmiuden yhteyden muodostamiseen. Lisäksi kolmitiekättelyllä varmistetaan verkkoolosuhteet ja niiltä osin soveltuvuus luotettavan TCP-yhteyden muodostamiselle. Kolmitiekättely myös pyrkii takaa-



Kuva 4.1: TCP SYN-ACK ja FIN-ACK

maan, että yhteyden molemmat osapuolet ovat todella sitä miksi esittäytyvät ja todella tarjoavat sitä palvelua, jota ollaan muodostamassa. Tämä suoritetaan sillä, että kättely on interaktiivinen toimenpide, jossa osapuolet joutuvat responsiivisesti reagoimaan toistensa viesteihin. Nimensä mukaisesti kolmitiekättelyssä on kolme vaihetta;

1. SYN, Yhteyden aloittaja, useimmiten asiakas, lähettää vastaanottajalle, useimmiten palvelimelle, SYN-viestin, jonka otsakkeen SYN-kenttä on aktivoituna
2. SYN+ACK, Palvelin vastaa omalla SYN-viestillään joka pitää sisällään järjestysnumeron. Lisäksi palvelin vahvistaa asiakkaan SYN-viestin
3. ACK, Asiakas vahvistaa palvelimen SYN-viestin.

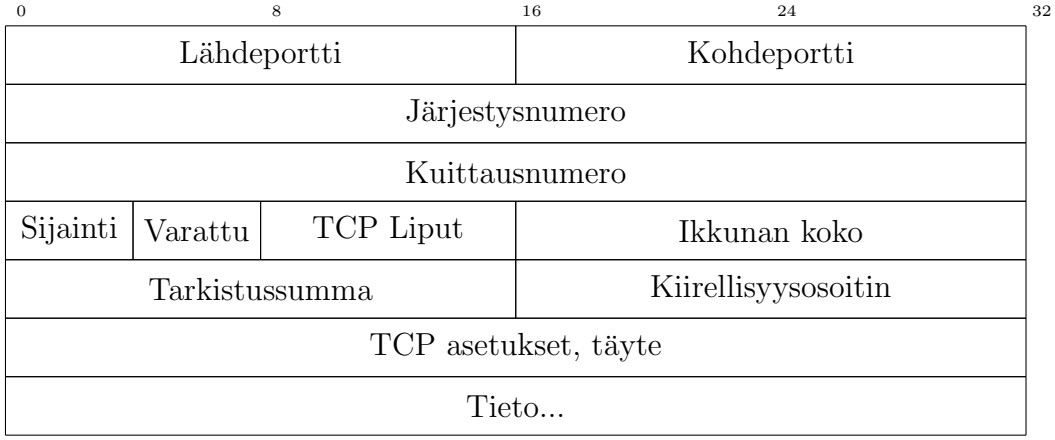
Lähtökohtaisesti TCP-yhteys avataan aina tällä mallilla, ja se vaatii siten vähintään tämän verran viestinvaihtoa. Viestinvaihto käydään joka kerta kun TCP-yhteys avataan. Siinä tapauksessa, että jokin näistä viesteistä hukkuu tai epäonnistuu, voi yhteyden avaaminen vaatia usemmankin viestin lähettämisen asiakkaan ja palvelimen välillä. TCP-protokolla on kuitenkin rakennettu niin ettei esimerkiksi viestin väliaikaisesta hukkumisesta aiheudu ongelmaa. Esimerkiksi jos asiakas lähettää viestin palvelimelle muttei saa siihen vastausta, ja päätyy lähettämään toisen viestin, joka johtaa tilanteeseen, että palvelin saakin lopulta kaksi kertaa täysin saman sisältöisen viestin ei se aiheuta ongelmaa TCP-protokollan toiminnan kannalta. TCP-protokolla tunnistaa viestit järjestysnumeron avulla, ja siten tunnistaa jos verkossa liikkuu kopioita samasta viestistä. Oletuksena TCP-protokolla hylkää ja tiputtaa verkkoliikenteestä pois sellaiset viestit, jotka on jo saavuttaneet kohteensa, mutta jostain syystä vielä saapuvat uudelleen.[22]

Yhteyden avaamisen jälkeen TCP-protokolla kuljettaa varsinaisia tietopaketteja asiakkaan ja palvelimen välillä. Pakettien lähettäminen jatkuu kunnes jompikumpi osapuoli toteaa saaneensa kaiken tarvittavan ja päättää lopettaa yhteyden. Protokollan mukaisesti yhteys lopetetaan FIN-ACK-menettelyllä, joka on hyvin samanlainen kuin yhteydenavausmekanismi. Nykyisessä verkkoliikenteessä on myös yleistynyt tapa jossa TCP-yhteydelle annetaan RST, eli RESET käsky, joka sulkee verkko yhteyden välittömästi. Käytännössä kuitenkin RST on alunperin tarkoitettu poikkeustilanteisiin, eikä varsinaisesti yhteyden päättämiseen. FIN-ACK-menettelyllä yhteys suljetaan yhteisymmärryksessä molempien osapuolien toimesta, eikä kummallekaan osapuolelle jää epäselväksi miksi yhteys loppui. Kuvassa 4.1 on kuvattuna myös TCP:n standardi tapa suorittaa FIN-ACK-menettelyllä yhteyden lopetus.

FIN-ACK menettely takaa sekä palvelimelle että asiakkaalle molempien osapuolten olevan valmiita kaiken tiedonsiirron osalta, ja valmiita siihen että yhteys suljetaan. [22]

4.2 Otsake

WWW-liikenteessä TCP käytännössä huolehtii siitä, että asiakaslaitteiden väliset HTTP-paketit kulkevat perille. TCP ei kuitenkaan varsinaisesti ota kantaa tietotyyppiin, jota se kuljettaa. TCP voi kuljettaa käytännössä mitä tahansa tietoa, siitä välittämättä mitä tieto todellisuudessa sisältää. Protokollan otsake on myös verrattain yksinkertainen ja suoraviivainen. Se pitää sisällään kuljetuskerroksen osalta tärkeän informaation, eikä juurikaan ylimääräistä. Kuitenkin kuten aiemmin todettu on otsakkeessa varattuna muutamia kenttiä mahdollisten lisäosien ja laajennuksien käytölle.



Kuva 4.2: TCP-otsake

Kuvassa 4.2 on kuvattuna TCP-protokollan otsake. TCP-otsakkeessa on aina vähintään kaksikymmentä tavua tietoa. Yhteyden käyttäessä joitain erityisiä TCP asetuksia voi tiedon määrä otsakkeessa olla jopa 23 tavua. Verrattuna esimerkiksi toisen siirtokerroksenprotokollan UDP:n otsakkeeseen, joka on keskimäärin noin kahdeksan tavua, on TCP:n otsake huomattavasti suurempi. Tämä ero tulee suurissa määrin nimenomaan TCP:n tarjoamasta turvasta tiedonsiirrossa, eli siis erilaisista varmistuksista ja tarkistuksista, joilla voidaan tiedon kulku taata yhteysvälillä. [22]

Kuten kuvasta 4.2 nähdään on verkkolaitteiden portit kuljetuskerroksenprotokollan kannalta merkittävä informaatio. Lähde- ja kohdeporttien kentät

ovat molemmat kahden tavun pituisia, mikä mahdollistaa 65535 eri porttia. Esimerkiksi HTTP-protokolla kulkee oletusarvoisesti TCP-protokollan portissa numero 80 tai 8080, mutta käytännössä ei ole mitään estettä sille etteikö HTTP voisi kulkea jossain toisessakin portissa. Varsinainen kohdeinformaatio verkossa kulkee TCP-protokollan alla olevan IP-protokollan kantamana, joten TCP:n ei tarvitse ottaa kantaa verkkoosoitteeseen millään tapaa. [22]

Järjestysnumero on merkittävä TCP-protokollan toiminnan kannalta. Järjestysnumeron avulla osiin jaetut (eng. fragmented) paketit saadaan kohteessa palautettua takaisin alkuperäiseen muotoonsa oikeassa järjestyksessä. Lisäksi järjestysnumeroa käytetään pakettien kopioiden tunnistamiseen, ja oletuksena TCP-protokolla pudottaa käsittelystä paketin, joka on täydellinen kopio jo käsitellystä paketista. Järjestysnumeron arvoa kasvatetaan yhteyden aikana lähetettyjen pakettien mukaan. Esimerkiksi jos yhteys olisi täysin vuoropuhelua kasvatettaisiin järjestysnumeroa vuorotellen. Otsakkeessa on yksi TCP-protokollan merkittävimmistä ominaisuuksista eli kuittausnumero. Kuittausnumeron avulla lähettäjä voi ilmaista odottavansa seuraavaksi saapuvaksi kyseisellä numerolla varustettua pakettia. Käytännössä kuittausnumeroa käytetään lähettäjän halutessa kuittauksen lähettämälleen paketille, ja esimerkiksi yhteydenmuodostustilanteessa kuittausnumeroa tulee käyttää aina.[22]

Kaksi tavua protokollassa on käytetty ilmaisemaan viestityyppiä ja varsinaisen tiedon sijaintia, eli käytännössä sitä kuinka pitkä vaihtuvamittainen otsake todellisuudessa on. Viestityyppejä TCP-protokollassa on kiireellinen-, kuitattava-, työntö toiminto-, uudelleenkäynnistys-, synkronoi järjestysnumerot- ja lopetusviesti. Jokaisella viestityypillä on oma tehtävänsä, mutta joitakin niistä voidaan käyttää myös rinnakkain samassa viestissä. Ikkunan koko -kenttä ilmaisee vastanottajalle kuinka monta tieto-oktettia lähettäjä on valmis vastaanottamaan paluuviestissä. Käytännössä ikkunan koolla kontrolloidaan nimenomaan lähetysnopeutta, ja esimerkiksi ruuhkanhallinnan kannalta ikkunan koko on olennainen kenttä TCP-otsakeessa.[22]

Tarkistussummaa käytetään nimensä mukaisesti paketin eheyden varmistamiseksi ja tarkistamiseksi. Tarkistussumma lasketaan koko paketin sisällöstä mukaan lukien otsake itsessään. Otsaketta laskettaessa täytyy huomioida, että tarkistussumma on riippuvainen myös tarkistussumma-kentän arvosta, minkä takia on sovittu, että tarkistussummaa laskettaessa tarkistussummakenttä merkataan aina nollaksi. Tarkistussumman avulla vastaanottaja voi varmistua että viesti on säilynyt muuttumattomana ja ehjänä lähetyksen läpi. Kiireellisyysosoittimella on tarkoitus antaa paketille käytännössä etuoikeus liikkua verkossa. Kiireellisyysosoitin osoittaa vastaanottajalle milloin

kiireellinen lähetys päättyy, käytännössä tämä tarkoittaa viimeistä tavua kiireellisessä lähetyksessä.

Lopuksi TCP-otsake pitää sisällään TCP-asetukset. TCP-asetuksia ei ole alkuperäisessä standardissa määritelty kovinkaan tarkkaan, vaikka niille on otsakkeesta varattu kohtuullisen paljon tilaa. TCP-asetukset kenttä voi kuitenkin olla täysin tyhjä. Nykyaikainen TCP-protokolla käytännössä kuitenkin sisältää aina joitakin asetuksia. TCP-protokollaa on täydennetty useilla kymmenillä, ellei jopa sadoilla RFC:eillä, joista monessa käytetään erityisesti asetuksetkenttää jonkinlaisen lisätoiminnallisuuden mahdollistamiseksi. TCP-asetukset onkin ollut merkittävä määrittely alkuperäisessä TCP-protokollassa takaamaan protokollan mahdollisuudet kehittyä ja muuttua verkkoliikenteen kasvaessa ja muuttuessa. TCP-asetukset lasketaan mukaan tarkistussummaan kokonaisuudessaan, joten tarkistussumman laskenta voidaan suorittaa vasta kun koko paketti on muilta osin jo määritelty.[22]

4.3 Ruuhkanhallinta

Ruuhkanhallinnalla tarkoitetaan TCP:n mekanismeja, joilla on tarkoitus vähentää liikenteen ruuhkautumista. Liikenteen ruuhkautuminen johtaa reitittimissä pakettien pudottamiseen. Pakettien pudottaminen taas lisää uudelleenlähetysten määrää, mikä heikentää liikenteen sujuvuutta kokonaisuutena.

TCP:n ruuhkanhallintamekanismi pyrkii reaktiivisesti välttämään ruuhkaa. Oletuksena ruuhkanhallintamekanismin havaitessa paketin hukkumisen se puolittaa koko TCP virran lähetysnopeuden. TCP-virran lähetysnopeutta kasvatetaan vähitellen kohti vastaanottajan ilmoitetun lähetysikkunan kokoa. Kasvattamista jatketaan kunnes ensimmäinen paketti hukkuu, mikä aiheuttaa jälleen lähetysnopeuden puolittamisen. Mekanismin luonteesta johtuen TCP-liikenne aiheuttaa verkon kuormitukselle luontaista huojuntaa, sillä liikennemäärä nousee jatkuvasti siihen saakka kunnes paketteja hukkuu ja liikenne puoliintuu.

Yhdessä HTTP-protokollan kanssa TCP:n ruuhkanhallinta toimii verrattain hyvin. HTTP-liikenne on tyypiltään vuoropuhelumaista, jolloin TCP:n tapa nostaa lähetysnopeutta vähitellen sopii HTTP:n liikennetyypille hyvin. HTTP-yhteydet yleensä alkavat asiakkaan esittämillä pyynnöillä, joissa viestien sisällöt on pienempiä kuin myöhemmin yhteyttä lähettävissä sivuston sisältöä sisältävissä paketeissa. Yleistynyt tapa avata rinnakkaisia HTTP-yhteyksiä asiakkaan ja palvelimen välille on aiheuttanut ongelmia TCP:n

ruuhkanhallinnan kannalta. Asiakkaan avatessa rinnakkaisia HTTP-yhteyksiä, ja siten myös rinnakkaisia TCP-yhteyksiä, tulee asiakas luoneeksi itselleen kilpailutilanteen. Rinnakkaiset TCP-yhteydet pitävät jokainen omaa ruuhkanhallintaansa tietämättään, että rinnalla kulkeva TCP-yhteys kantaa käytännössä saman HTTP-istunnon pyyntöä. Tämä luo tilanteen, jossa rinnakkaiset TCP-yhteydet kilvan nostavat lähetysnopeuttaan, ja siten syövät toisiltaan käytössä olevaa verkkoresurssia.

TCP-protokollan on mahdollista käyttää vaihtoehtoisia algoritmeja ruuhkanhallintaan. TCP:lle esitettyjä ruuhkahallintaalgoritmeja on olemassa lukuisia erilaisia, ja niiden tarkoituksena on tarjota soveltuvaa ruuhkanhallintaa erityyppisille verkkoliikenteen muodoille, kuten WWW-sivut, tiedostojen kopiointi ja suoratoistopalvelut. Tämä on ehdottamasti etu WWW-liikenteenkin kannalta.

Esimerkiksi Torrent-liikenteelle on kehitetty oma algoritminsa, Low Extra Delay Background Transport (LEBBAT), jonka tarkoituksena on väistää käytännössä kaikkea muuta liikennettä, joka käyttää samoja verkkoresursseja. Tämä tarkoittaa WWW-liikenteen kannalta sitä, että hidas tiedonsiirto tässä tapauksessa väistäisi ja antaisi tilaa WWW-liikenteelle, joka taas takaisi ruuhkaisilla verkkoresursseilla paremman suorituskyvyn WWW-liikenteelle kuin normaalissa kilpailutilanteessa. [18]

Vastaavasti esimerkiksi TCP NewReno-algoritmi muokkaa TCP:n oletuksena toimivaa algoritmia niin, että sen suurimmat heikkoudet lievenisivät. NewReno toimii oletusalgoritmia rauhallisemmin lähetysnopeutta säädellessään. NewReno ei välittömästi yhden paketin tiputtua puolita lähetysnopeutta, vaan odottaa hieman pidmepään. NewReno on ollut yksi merkittävimpiä uudistuksia TCP-protokollaan, sillä se on parantanut merkittävästi verkkojen siirtokykyä yksittäisten pakettien hukkuessa. NewReno ei kuitenkaan ole täysin optimaalinen ratkaisu verkkoihin, joissa paketteja hukkuu useita kerrallaan tai verkkoihin, joissa paketteja hukkuu jatkuvasti. [16]

Mahdollisuus ruuhkanhallinnan algoritmin vaihteluun ja kehittämiseen on loistava ominaisuus TCP-protokollassa, sen avulla protokollan on mahdollista kehittyä ja muuttua esimerkiksi WWW-liikenteen tarpeiden mukaisesti. Algoritmin vaihtamisen mahdollisuus myös antaa TCP-protokollalle mahdollisuuden mukautua käytännössä minkä tahansa verkkoliikenteen tarpeisiin, kuten LEDBAT-esimerkki osoittaa. Tämä monipuolistaa TCP-protokollan toimintamuotoja.

4.4 TCP:n aiheuttamat viiveet kuljetuskerroksella

Tässä osiossa käydään läpi ominaisuuksia ja mekanismeja, jotka osaltaan tuottavat tarpeetonta viivettä nykyiseen WWW-liikenteeseen. TCP:n synnyttämä viive aiheutuu, joko suoraan viivettä synnyttävistä mekanismeista ja välillisesti viivettä synnyttävistä ja aiheuttavista mekanismeista. TCP-protokollaan suoraan viivettä aiheuttaa rakenteelliset ongelmat, joiden takia protokolla jo lähtökohtaisesti aiheuttaa tai voi aiheuttaa viivettä ja pidentää pakettien käsittelyaikoja. Tälläisiä ominaisuuksia ovat esimerkiksi kolmitiekättely, uudelleen lähettely ja järjestysnumerointi. Kolmitiekättely on ymmärrettävästi hidastava tekijä. Kolmitiekättelyllä on puolensa, mutta kiistämättä on selvää, että kolmen paketin lähettäminen ennen yhteyden aloittamista on hidastava tekijä. Vastaavat ominaisuudet ja hyödyt voidaan saavuttaa ilmankin raskasta kolmitiekättelyä, tai ainakin ilman, että se suoritetaan joka kerta asiakkaan ja palvelimen välillä kun avataan uusi TCP-yhteys.

TCP:hen kehitetty TCP Fast Open -laajennos, jonka avulla yhteyden avauksessa lähettäminen saadaan tapahtumaan tavallista aiemmin[7]. TCP Fast Open:in toiminta käsitellään tarkemmin luvussa 4.5. Kolmitiekättelyn raskaus korostuu yhteyksissä joissa itse lähettävän tiedonmäärä on pieni ja täten avattava yhteys pysyy yllä vain lyhyen aikaa. Lyhyen yhteyden yleiskustannukset, eli protokollan vaatimat kustannukset, nousevat todella suureen osaan koko yhteyden tiedon määrästä. Kolmitiekättelyä nopeuttavaa prosessia käsitellään tarkemmin luvussa 4.5.

Toinen TCP:n kannalta ongelmallinen ja pahimmassa tapauksessa viivettä aiheuttava tekijä on järjestysnumerointi paketeissa. Järjestesnumerointi on tärkeä ominaisuus TCP:n yleisen toimintaperiaatteen kannalta, sillä se takaa tiedon toimittamisen luotettavasti. Tiedon toimittaminen luotettavasti tarkoittaa tiedon toimittamista oikeassa järjestyksessä ilman aukkoja tiedonkulussa. Tämä aiheuttaa ongelman siinä tapauksessa, että käsittelyssä olevan paketin käsittelyyn tulee jonkinlainen viive tai muu ongelmatilanne. Esimerkiksi jos lähetetty paketti hukkuu matkalla ja joudutaan protokollan mukaisesti uudelleenlähettämään puuttuva paketti, niin käytännössä muut paketit odottavat käsittelyä siihen saakka kunnes samasta jonosta puuttuva paketti on saatu uudelleenlähetettyä. Tämä on ongelmallista sillä välttämättä seuraavat paketit eivät sisällä mitään sellaista minkä takia niiden käsittelyn olisi odottetta edellistä pakettia. Toisaaltaan myös moni paketti voitaisiin käsitellä, tai ainakin tarkistaa, että voidaanko ne käsitellä, ilman odotettavaa

pakettia. Tällä hetkellä kuitenkin odottettava paketti käytännössä pysäyttää pakettien käsittelyn hetkellisesti. Lisäksi tälläinen toiminta aiheuttaa sen, että vastaanottajan välimuisti täyttyy jonossa olevista paketeista, jotka odottavat käsittelyä. Tämän tyyppistä ongelmaa kutsutaan yleisesti verkkoliikenteessä termillä "Head-of-Line blocking", eikä esiinny ainoastaan TCP-liikenteen tapauksessa vaan yleisemminkin WWW-liikenteessä. TCP:n tapauksessa ongelma liittyy myös siihen ettei TCP varsinaisesti tue liikenteen limittämistä. Limityksen mahdollistaminen keinotekoisesti aiheuttaa sen, että pahimmillaan yhden paketin hukkuminen aiheuttaa liikenteelle viivettä koko sovelluskerroksen protokollan osalta. Tämä oli yksi suurimpia motivaatioita Googlella sen aloittaessa kehittää vaihtoehtoa TCP-protokollalle. Tätä vaihtoehtoa käsitellään tarkemmin luvussa 5.

4.5 TCP:n ominaisuudet WWW-liikenteelle

Kuten tämän luvun alussa todettiin, on TCP:n ominaisuuksia monipuolistettu lisäosilla ja laajennuksilla. Osa näistä lisäosista ja laajennuksista on tehty alkujaan nimenomaan WWW-liikennettä varten.

Kolmitiekättelyn hitaus on ollut eräänlainen ongelma TCP-liikenteessä, ja esimerkiksi WWW-liikenteessä sen tuottama viive voi olla hyvinkin merkittävää pienten sivustojen avaamisen yhteydessä. TCP Fast Open -tekniikan avulla kolmitiekättelyä voidaan nopeuttaa, tai oikeastaan lähetyksen aloittamista voidaan nopeuttaa kolmitiekättelystä huolimatta. TCP Fast Open on verrattain uusi tekniikka, se on esitelty vasta vuoden 2014 lopussa, joten käytännössä sen käyttöönotto on tapahtunut vuoden 2015 aikana. Ajatuksena TCP Fast Openissa on tallettaa asiakkaan puolelle tieto siitä, että palvelin on kertaalleen tutustunut asiakkaaseen ja siten kolmitiekättely voidaan jatkossa suorittaa nopeammin. Käytännössä tämä suoritetaan tallettamalla salakirjoitettu eväste asiakkaan puolelle. Eväste lähetetään asiakkaalle palvelimen toimesta ensimmäisellä kerralla kun nämä kaksi osapuolta luovat yhteyden. Eväste siirretään palvelimelta asiakkaalle TCP-otsakkeen asetukset kentässä, joka on esitetty kuvassa 4.2. Asiakkaalle tallennettu eväste auttaa asiakasta nopeuttamaan uuden yhteyden luomista samaan palvelimeen. Uuden yhteyden avauksen yhteydessä asiakas lähettää SYN-paketin yhteydessä muistiinsa tallentaman evästeen, jonka avulla tunnistautuu palvelimelle. Palvelimen tunnistaessa, että kyseessä on jo aiemmin asioinut asiakas, voi palvelin aloittaa lähettämisen jo ennen kuin asiakkaalta on saapunut kolmitiekättelyn vahvistava ACK-paketti. Tälläisellä menettelyllä palvelin säästää yhteyden odottelussa yhden kokonaisen kierroksen asiakkaan ja pal-

velimen välillä, ja näin ollen vähentää merkittävästi ennen varsinaista tiedon lähetystä edeltävää viivettä.[7]

Fast Open:in lisäksi TCP-protokollaan on ehdotettu lähetysikkunan suurennusta jo yhteyden alkaessa. Oletuksena TCP:n lähetysikkunan koko on ollut yhdestä neljään segmettiä, yleisimmin yksi tai kaksi segementtiä. Uudessa RFC:ssä ehdotetaan lähetysikkunan kasvattamista jo yhteyden alkaessa kymmeneen segmenttiin. Kasvattamista perustellaan sillä, että nykyiset verkkoyhteydet ovat aiempaa parempia, ja niiden kapasiteetti riittää useimmissa tilanteissa suurempaa ikkunakokoon. Verkkojen kapasiteetin arvellaan riittävän suurempaan ikkunakokoon siitäkin huolimatta, että asiakkaalla olisi jo valmiiksi yhteyksiä käynnissä. RFC kuitenkin huomauttaa, että useiden rinnakkaisten yhteyksien määrä voi ennen pitkää johtaa tilanteeseen, jossa kapasiteetti loppuu kesken. Käytännössä tämä ongelma syntyy kuitenkin enemmänkin rinnakkaisten yhteyksien lukumäärästä kuin lähetysikkunan koosta. Lähetysikkunan avaaminen suurempana heti yhteyden alkaessa parantaa yhteyden lähetysnopeutta, sillä normaalisti TCP-protokollalla kestää useamman RTT:n ajan saavuttaa vastaava lähetysnopeus. Tämä korostuu lyhytkestoisissa yhteyksissä, joissa ei välttämättä muuten saavutettaisi vastaavaa lähetysnopeutta koskaan yhteyden lyhytkestoisuuden takia. Ominaisuus saattaa siis parhaassa tapauksessa parantaa sekä yksittäisen yhteyden suorituskykyä että koko verkon suorituskykyä. [6]

Vuoden 2013 SIGCOMM:ssa Google on esittänyt useita mielenkiintoisia menetelmiä, joiden avulla voitaisiin nopeuttaa ja parantaa asiakaskokemusta. SIGCOMM on vuotuinen johtava verkkotekniikan tapahtuma.[12] Osa esitellyistä muutoksista koskee myös TCP-protokollaa. Artikkelissa todetaan, että TCP-protokollan suorituskykyä voitaisiin erityisesti parantaa virhetilanteissa, joissa paketteja hukkuu. Nykyisellään virhetilanteesta palautuminen on hidasta ja aiheuttaa merkittävää haittaa yhteyden suorituskyvylle. Google esittelee useita tekniikoita, joiden avulla virhetilanteista palautumista voitaisiin nopeuttaa. Googlen mallissa erilaisia tekniikoita käytettäisiin välityspalvelimien ja asiakkaiden välillä, ja taas toisaaltaan välityspalvelimen ja varsinaisen palvelimen välillä. Välityspalvelimen eri puolilla on tarpeen käyttää erilaisia tekniikoita, sillä verkkoyhteydet ovat täysin erilaiset. Yleensä välityspalvelimen ja varsinaisen palvelimen välillä on redundanttinen yhteys, joka ei juurikaan aiheuta pakettien hukkumista. Välityspalvelimen ja asiakkaan välillä taas saattaa olla hyvinkin vaihtelevia yhteyksiä, mutta niille kaikille yhteistä on yleensä verrattain suuri pakettien hukkumisprosentti ja melko lyhyt verkon synnyttämä viive asiakkaalle. Tämä ero olosuhteissa aiheuttaa täysin erilaiset tarpeet virhetilanteesta toipumiselle, ja sen takia Google esittääkin eri tekniikoita. Käytännössä Google esittää asiakkaan ja

välityspalvelimen välille reaktiivista tekniikkaa, joka uudelleenlähettää paketin tietyn odotusajan jälkeen, kuitenkin ennen kuin asiakas on varsinaisesti pyytänyt uudelleenlähetystä. Google esittää, että erityisesti lähetysten viimeiset paketit lähetettäisiin uudelleenlähetyksenä reaktiivisesti. Tällä toiminnalla nopeutetaan prosessia entisestään. Asiakkaan saadessa lähetyksen viimeinen paketti, saa asiakas käytännössä tietoonsa myös mahdolliset muut paketit jotka siltä puuttuvat, ja näin ollen asiakas pystyy lähettämään palvelimelle pyynnön puuttuvista paketeista.

Sen sijaan välityspalvelimen ja varsinaisen palvelimen välille Google esittää proaktiivista lähetysmismallia, jossa käytännössä kaikki pakettienlähetykset tuplataan. Lähetysliikenne on verkkaisempaa varsinaisen palvelimen ja välityspalvelimen välillä, sillä muutoksia verkkosivulle ei käytännössä suoriteta jatkuvasti vaan välityspalvelimen välimuistia päivitetään ajoittain. Proaktiivisessa esityksessä lähetetään jokaisesta paketista käytännössä heti perään duplikaatti, eli kopio, joka on käytännössä täysin ylimääräistä tietoa normaalissa tilanteessa. Tämä lisätty verkkoliikenne ei kuitenkaan vaikuta kyseisessä verkossa sillä kapasiteetti on käytännössä rajaton, eikä jokaisen paketin tuplaaminen käytännössä vaikuta suorituskykyyn. Lisäksi TCP-protokollan mukaan normaalitilanteessa ylimääräiset paketit vain jätetään huomioimatta. Tällä tavalla verkossa, jossa pakettien hukkuminen on harvinaista, saavutetaan parempi suorituskyky, sillä kahden paketin hukkuminen peräkkäin on vieläkin harvainaisempaa. Paketin hukkuessa välityspalvelin käyttää automaattisesti saamaansa kopiota, eikä tilanne aiheuta minkäänlaista uudelleenlähettämisprosessia.[12]

Lisäksi Google ehdottaa FEC-ominaisuuden lisäämistä TCP-protokollaan. Tämä lisää jonkin verran lähetettävän tiedon määrää, mutta Googlen laskelmien mukaan FEC-ominaisuus kuitenkin aiheuttaa vähemmän viivettä kuin uudelleenlähettäminen. Uudelleenlähettäminen koetaan olevan suuri ongelma TCP-protokollan kannalta. Syitä uudelleenlähettämisen ongelmiin on käsitelty tämän työn luvussa 4.4. FEC-ominaisuus mahdollistaa asiakkaan itsenäisesti ratkaista puuttuvan paketin sisältö saamiensa pakettien perusteella. FEC-korjaus kuitenkin käytännössä koskisi vain tapausta, jossa yksi paketti on hukkunut. Google on kuitenkin esityksessään todennut, että useimmiten asiakkailla hukkuu yksittäisiä paketteja, jotka sijouttuvat lähetyksen loppuvaiheeseen, eli käytännössä lähetyksen viimeiseen tai toiseksi viimeiseen pakettiin. [12]

Luku 5

Quick UDP Internet Connection - QUIC

Quick UPD Internet Connection (QUIC) on suunniteltu yhdessä HTTP/2-protokollan kanssa. QUIC:in tehtävänä on toimia kuljetuskerroksen ja sovelluskerroksen välillä toimivana protokollana. QUIC:in tavoitteena on kuitenkin korvata toiminnallaan TCP-protokolla ja TLS-protokolla. Nimensä mukaisesti QUIC käyttää toimiessaan UDP-protokollaa. QUIC:in tarkoituksena tarjota kuljetuskerroksella tarvittavia mekanismeja ylemmän tason protokollien käyttöön. QUIC on tarkoitettu erityisesti toimivaksi HTTP/2-protokollan kanssa, ja sen RFC:kin on nimetty sen mukaisesti: "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2". Käytännössä kuitenkaan mikään ei estä sitä, että QUIC:iä ei voisi käyttää minkä tahansa muunkin datan siirtoon. QUIC on kuitenkin erityisesti optimoitu käytettäväksi yhdessä HTTP/2-protokollan kanssa. QUIC:in luvataan tarjoavan käytännössä samat toiminnallisuudet kuin TCP:nkin, mutta käyttäen UDP-siirtoprotokollaa. UDP on siirtokerroksen protokolla, joka TCP:n ohella tarjoaa siirtokerroksen palveluja. UDP-protokolla ei ole luotettava protokolla, joten se ei varmista eikä takaa lähettäessään tiedon kulkevan perille saakka ehjänä. QUIC:ssä on päästä päähän suojattu yhteys jonka luvataan olevan TCP:n TLS:ää vastaava. Lisäksi toisin kuin UDP-yhteydet yleensä QUIC tarjoaa luotettavan päästä päähän yhteyden ja ruuhkanhallintamekanismit samoin kuin TCP-protokolla. QUIC pyrkii rakentamaan luotettavan protokollan ominaisuudet UDP-protokollan ympäristöön, jossa lähtökohtaisesti siirtonopeus on priorisoitu luotettavuuden edelle. QUIC on aktiivisessa kehityksessä, sen protokolla spesifikaatiota on päivitetty kolme kertaa vuoden 2015 aikana. Selaimista erityisesti Chromium-selaimen mainitaan käyttävän QUIC toiminnallisuutta ja kehittäjät keräävät jatkuvasti kokemuksia nykyisen standardin toimivuu-

desta ja heikkouksista. [15]

5.1 Protokollan toiminta

QUIC pyrkii ratkaisemaan useita nykyisen WWW-liikenteen kuljetus- ja sovelluskerroksen ongelmia. QUIC:in keskeisimmät ominaisuudet, joita se on pyrkinyt kehittämään WWW-liikenteen eduksi ovat yhteyden muodostamisessa syntyvät viiveet, ruuhkanhallinta, pakettien lomittaminen, pakettien virheenkorjaus ja yhteyksien siirrot. QUIC:in pyrkimyksenä on esimerkiksi yhteyden muodostamisessa parhaimmassa tapauksessa päästä tilanteeseen, jossa yhteyden muodostamista ei käytännössä tarvita, vaan asiakas voi aloittaa lähettämisen käytännössä suoraan. Suoraan lähettäminen yksinkertaistettuna mahdollistetaan asiakkaalle ja palvelimelle tallennetuilla evästeiden kaltaisilla tiedoilla, jotka mahdollistavat päätepisteiden tunnistamisen välittömästi yhteyden alkaessa. Toimintamalli on käytännössä hyvin samanlainen kuin TCP:n Fast Open-tekniikan toiminta. TCP:n Fast Open on käsitelty tarkemmin luvussa 4.5. Huonoimmassakin tapauksesssa QUIC-protokolla tähtää muodostavansa yhteyden nopeammin tai vähintään yhtä nopeasti kuin TCP-protokollan kolmitiekättely.[1]

QUIC uudistaa kuljetuskerroksen siirtotekniikan vuoropohjaisesta keskustelusta viestivirraksi. Viestivirta mahdollistaa aiempaa paremman mahdollisuuden yhteyden tarkkailuun ja sen myötä ruuhkanhallintaan. Viestivirran avulla erityisesti lähetysnopeuden säätely helpottuu, sillä viestivirta kohtaisesti tapahtuvan monitoroinnin avulla vastaanottaja voi paremmin määrittää lähettäjälle haluamansa nopeuden. Jokainen QUIC-viesti sisältään uuden järjestysnumeron ja aikaleiman, mikä parantaa QUIC:n ruuhkanhallintaan liittyvien algoritmien toimintaa. Esimerkiksi TCP:ssä uudelleenlähetetyt paketit saavat alkuperäisen paketin järjestysnumeron, koska TCP:n tapa luotettavan yhteyden toimintaan vaatii sitä. Tämä kuitenkin hankaloittaa ruuhkanhallintaan, sillä yksittäisen paketin virtaamista on hankalampi monitoroida kun virrasta löytyy pahimmassa tapauksessa useita jäljennöksiä samasta paketista.

Viestivirta tuo tulleessaan QUIC:iin viestivirtatunnisteen. Tunnisteen avulla QUIC parantaa merkittävästi nykyisten yhteyksien siirtoja. Yhteyksien siirrolla tarkoitetaan tilannetta, jossa käyttäjä siirtyy esimerkiksi WLAN-verkossa tukiasemalta toiselle, ja tämä siirtyminen aikaan saa käyttäjällä IP-osoitteen vaihdoksen. TCP-protokollassa IP-osoitteen muuttuminen tarkoittaa sitä, että olemassa olevat TCP-yhteyden katkeavat ja yhteydet täytyy

muodostaa uudelleen. QUIC:in yhteystunniste sen sijaan mahdollistaa sen, että IP-osoitteen muuttumisesta huolimatta asiakas voisi jatkaa samojen yhteyksien käyttöä, sillä palvelin tunnistaa asiakkaan viestivirtatunnisteen pohjalta, eikä IP-osoitteen. Viestivirta mahdollistaa QUIC:iin myös aikaisempaa paremman mahdollisuuden lomittamiselle. QUIC:in viestivirrat ovat itsenäisiä kokonaisuuksia asiakkaan ja palvelimen välillä. Lomittaminen onnistuu rinnakkaisia viestivirtoja avaamalla luonnollisesti ja helposti. Tämä mahdollistaa sen, etteivät rinnakkaiset viestivirrat häiriinny toistensa mahdollisista ongelmista, vaan asiakas ja palvelin voivat tehokkaasti siirtää useaa eri kokonaisuutta rinnakkain. Pakettien hukkuminen yhdessä viestivirrassa ei vaikuta muiden viestivirtojen toimintaan millään tavalla. Seuraavissa luvuissa tutustutaan paremmin niihin keinoihin, joilla QUIC pyrkii edellä mainitut ominaisuudet toteuttamaan. Tämän lisäksi käydään läpi QUIC-protokollan pakettityypit, kehykset ja otsake. [15]

5.2 QUIC-pakettityypit

QUIC käyttää tällä hetkellä neljää erilaista pakettityyppiä. Standardissa pakettityypit jaetaan kahteen ryhmään, erikoispaketit ja yleiset paketit. Erikoispaketteja ovat Version Negotiation ja Public Reset -paketit. Yleisiin paketteihin kuuluvat Frame-paketit ja FEC-paketit. QUIC-paketin koko on vaihteleva, mutta pyrkimyksenä protokollalla on, että kaikki QUIC-paketit mahtuisivat yhteyden MTU:n sisään. Yhteyden MTU:n sisään mahduttamisella vältyttäisiin siltä, että IP-protokolla pilkkoo QUIC-paketteja useisiin IP-paketteihin. QUIC:in kehittäjät mainitsevat työskentelevänsä yhteyden MTU:n tunnistamisen kanssa[15]. Tällä hetkellä QUIC käyttää kiinteitä maksimi MTU-arvoja IPv6:lle ja IPv4:lle. QUIC-spesifikaatiossa varsinkin pakettityyppien kuvauksien kohdalla lukee useassa kohtaa ”TODO”, lisäksi protokolla spesifikaatioiden 01 ja 02 välillä pakettitoiminnallisuuksien kuvaukset ja pakettien datakentätkin ovat muuttuneet merkittävästi. Tämä käytännössä kertoo siitä, että QUIC-protokolla on vielä vahvasti kehityksen alainen, vaikka se onkin jo käytössä Googlen Chrome selaimessa. [14, 15]

QUIC:n yleiset paketit ovat salattuja ja todennettuja. Todennus tapahtuu yleisessä otsakkeessa olevien tietojen(pakettinumero, yhteys-id) avulla. Salaus tapahtuu välittömästi QUIC:n yhteisen otsakkeen jälkeen. Yhteisen otsakkeen jälkeen tulee välittömästi AEAD (Authenticated Encryption and Associated Data) tiedot. AEAD tiedosta täytyy purkaa salaus, jonka jälkeen on mahdollista ymmärtää paketin oikea sisältö. Kaikilla yleisillä paketeilla on yhteinen yksityinen otsake. Yleisten pakettien yksityinen otsake sisältää

Private Flags ja FEC -kentät, jotka molemmat ovat yhden tavun pituisia.

Yksityisille asetuksille (eng. privat flags) on varattu yksi tavu. Tavun ensimmäinen bitti on FLAG_ENTROPY bitti, joka on tarkoitettu tietopaketeille. Entropy-bitillä ilmaistaan, että paketti sisältään yhden bitin varmistuksen FEC-paketeille. Yksityisten asetusten toinen ja kolmas bitti ilmoittaa onko FEC-suojaus käytössä. Neljä seuraavaa bittiä yksityisissä asetuksissa ilmaisee, että käsittelyssä oleva paketti sisältään FEC-paketin. Jälkimmäinen tavu QUIC-paketissa on FEC-kenttä. FEC-kenttä sisältää tiedon FEC-suojaukseen kuuluvan pakettiryhmän ensimmäisestä paketista. FEC-kentässä on kuvattuna pakettinumero, joka viittaa FEC-ryhmäsuojauksen ensimmäiseen pakettiin. FEC-kenttä on läsnä paketissa vain, jos yksityisissä asetuksissa on määritetty, että paketti sisältää ylipäänsä FEC-suojauksen. [15]

Yksityisten asetusten jälkeen yleiset paketit sisältävät joko Frame-paketin tai FEC-paketin. Frame-paketissa on QUIC-protokollan varsinaiset tietokentät, joita QUIC:ssä nimitetään kehyksiksi. Frame-paketti koostuu toistuvasta määrästä kehyksiä, jotka koostuvat kahdesta kentästä, kehystyypistä ja kehyksen sisällöstä(eng. payload). FEC-paketti sisältää yksinkertaisen XOR-koodin jokaisesta FEC-suojatusta paketista.

QUIC:n erikoispaketit on suunniteltu käytettäväksi QUIC-yhteyden tilanteissa, jotka tapahtuvat käytännössä vain kerran yhden yhteyden aikana. Version Negotiation -paketti on tarkoitettu ainoastaan palvelimen lähetettäväksi ja Public Reset -paketti on tarkoitettu tilanteeseen, jossa joko palvelin tai asiakas haluaa katkaista yhteyden käytännössä välittömästi. Version Negotiation -paketin palvelin lähettää ilmoittaakseen tukemistaan QUIC-versioista. Paketti koostuu julkisista asetuksista ja yhteys-id:stä, joita seuraa lista palvelimella tuetuista QUIC-versioista. QUIC-versiokentät ovat 32-bittisiä. QUIC-versiokuvaus on sama kuin julkisessa otsakkeessa kuvattu QUIC-versiokenttä. Public Reset -paketissa on tarvittavat tiedot vastakkaiselle osapuolelle yhteyden katkaisemista varten. Käytännössä paketissa on Public Reset -ilmoitus, sekä vastakkaisen osapuolen viimeisimmäksi lähetetyn paketin pakettinumero. Viimeisen viestin pakettinumero on mukana siksi, että vastapuoli näin ilmoittaa jättävänsä kyseisen paketin käsittelemättä. QUIC:n Public Reset -toiminto vastaa toiminnollisesti TCP:n TCP-RST-toimintoa. [15, 19]

5.3 Kehykset

Kehykset ovat QUIC-protokollan ydin. Kehyksien avulla kuljetetaan ja hallitaan sovelluskerroksen protokollien tuottamaa liikennettä. Kehykset luovat

Frame-pakettien sisälle täysin uuden erillisen pakettijärjestelmän. Kehyksillä on omat otsakkeensa, joista käytännössä jokainen on erilainen kehysviestityypistä riippuen. Erilaisia kehysviestityyppejä QUIC Frame-paketin sisällä on tällä hetkellä 11 kappaletta ja ne jakaantuvat QUIC-pakettien tapaan erikoiskehystyyppeihin ja yleisiin kehystyyppeihin. Erikoiskehystyyppejä ovat STREAM-, ACK- ja CONGESTION_FEEDBACK-kehykset[15]. QUIC-protokollakuvaus rajoittaa kehykset yhteen pakettiin. Kehysten rajoittaminen yhteen pakettiin estää QUIC-protokollan liikenteen osittumisen (eng. fragmentation). QUIC-kehykset tarjoavat paljon TCP-protokollasta tuttuja toiminnallisuuksia. TCP:stä tuttuja kehystyyppejä ovat esimerkiksi RST_STREAM ja WINDOW_UPDATE. Seuraavassa käsitellään syvemmin joitakin QUIC-protokollan kehyksiä sekä esitellään niiden ominaisuuksia.

Stream-kehys on QUIC-protokollan ydin. Stream-kehys luo ja alustaa viestivirran. Luomisen ja alustamisen lisäksi se toimii viestivirran kuljettavana kehyksenä. Stream-kehyksen otsake koostuu kehystyyppikentästä, viestivirran-id:stä ja mahdollisisti kuormakentistä. Kuormakentät ovat läsnä kehyksessä ainoastaan silloin kun kehyksellä on kuljettavanaan jotakin kuormaa, eli käytännössä viestivirtaa luotaessa kuormakentät eivät ole kehyksessä. Kehystyyppikenttä toimii Stream-kehyksen sisäisenä otsakkeena. Kehystyyppikenttä on kahdeksan bittiä pitkä, joista jokainen on oma lippunsa tietylle toiminnolle. Osa lipuista ilmaisee kehystyyppikenttää seuraavien kenttien olemassaoloa tai pituutta, mutta niiden lisäksi kehystyyppikenttä ilmaisee onko kyseinen kehys viestivirran viimeinen kehys lähettäjän puolesta. Ilmaisemalla viestivirran viimeinen kehys, saatetaan yhteys puoliksi suljettuun tilaan, jossa lähettäjä on ilmaissut vastaanottajalle olevansa valmis lopettamaan viestivirran. Kehysviestissä täytyy aina olla jonkin verran hyötykuormaa, ellei kyseessä ole viestivirran viimeinen kehys, jolloin kehystyyppikenttä ilmaisee ainoastaan olevansa viimeinen kehys.

QUIC-kehykset käyttää myös ACK-kehystä, jonka tarkoituksena on kuitata toiselle osapuolelle jonkin lähetyksen onnistuneesti vastaanotetuksi. QUIC-protokollan ACK-kehykset viestivät käytännössä suurimman pakettinumeron, joka on vastaanotettu. Viimeisimmän vastaanotetun paketin pakettinumeron ohella ACK-kehys pitää sisällään listan puuttivien pakettien pakettinumeroista, eli siis paketeista joiden pakettinumerot ovat viimeisimmän paketin ja sitä edeltävän tiedossa olevan paketin välillä. ACK-kehyksen lisäksi toinen osapuoli lähettää säännöllisesti STOP_WAITING-kehystä, jonka tarkoituksena on ilmaista vastaanottajalle ettei tietystä pakettinumerosta alaspäin olevia pakettaja tulla koskaan uudelleen lähettämään tai ole koskaan lähetettykään, joten vastapuolen on turha niitä odottaa. Tällä toiminnolla saadaan rajattua ACK-kehyksen sisältämää listaa puuttuvista paketeista, sillä

STOP_WAITING siivoaa listalta ne paketit, joita lähettäjä ei aiokaan enää toimittaa. STOP_WAITING-kehyksellä on tarkoitus pienentää päätelaitteiden tarvetta pitää pakettinumeroita muistisssa ja siten vapauttaa resursseja käyttöön.

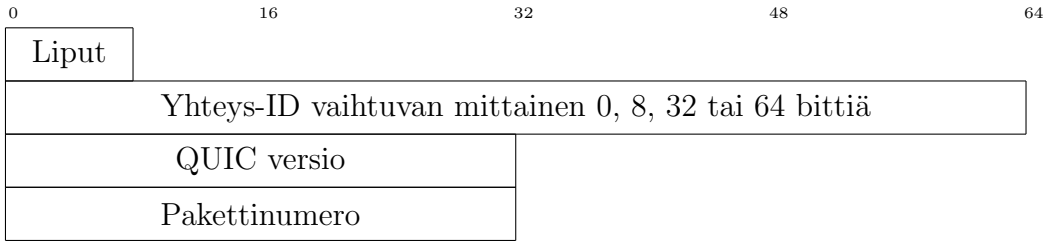
Yhteyden päättäminen tapahtuu CONNECTION_CLOSE-kehyksellä. CONNECTION_CLOSE-kehys on ilmoitus vastaanottajalle, että yhteys suljetaan nyt. Sulkemisilmoitus CONNECTION_CLOSE-kehyksellä on ehdoton. Kehyksellä kaikki olemassa olevat viestivirrat suljetaan välittömästi samalla kun yhteys katkaistaan. Käytännössä QUIC-protokolla ohjeistaa käyttämään GO_AWAY-kehystä ennen yhteyden implisiittistä sulkemista. GO_AWAY-kehys on suunniteltu palvelimelle tavaksi ilmoittaa asiakkailleen, että palvelin on menossa huoltoon ja olemassa olevat viestivirrat tulisi pyrkiä saattamaan loppuun tietyn aikarajan sisällä. Protokolla ohjeistaa kuitenkin kaikkia yhteyksiä ennen sulkemista ilmoittamaan yhteyden sulusta ennalta GO_AWAY-kehyksellä. On kuitenkin mahdollista, että käytännön toteutukset ohittavat tämän suosituksen ja sulkevat yhteydet tarvittaessa suoraan CONNECTION_CLOSE-kehyksellä, ilman "ylimääräistä" GO_AWAY-toimintoa. Näiden lisäksi QUIC:issa on RST_STREAM-kehys, jonka tarkoituksena on ilmaista osapuolen joutuneen poikkeavaan tilanteeseen ja sen myötä haluavan luopua viestivirrasta. RST_STREAM-kehystä tulisi siis käyttää ainoastaan virhetilan sattuessa ja sen tarkoituksena on vain tehdä yhteyden sulkeminen sovinnollisesti neuvotellen sen sijaan, että yhteys vain katkeaisi ja toinen osapuoli jäisi odottamaan.

Muut QUIC-protokollan kehystyypit ovat tyypiltään enemmän informatiivisiä ja niiden tarkoituksena on joko tarjota jonkinlaisia vianetsintää helpottavia toimintoja tai varmistaa yhteyden toiminta yhteysparametreja päivittäen. QUIC-protokollan muista kehystyypeistä mielenkiintoisena kehystyyppinä mainittakoon CONGESTION_FEEDBACK, jonka ilmoitetaan toistaiseksi olevan kokeiluvaiheessa. CONGESTION_FEEDBACK-kehys on tarkoitettu täysin ACK-kehysvirran ulkopuoliseksi toiminnoksi, tuottamaan tietoa ruuhkan syntymisestä ja kehittymisestä viestivirrassa. Tässä vaiheessa ei kuitenkaan ole tarkemmin kuvattu, millä tapaa CONGESTION_FEEDBACK-kehys tulisi toimimaan.[15]

5.4 QUIC-otsake

Kaikilla QUIC-paketeilla on yhteinen otsake, joka sisällytetään jokaisen paketin alkuun. Yhteisen otsakkeen jälkeen QUIC-paketeilla on paketti kohtai-

set tietosisällöt (eng. payload). Yhteinen otsake on vaihtuvamittainen, mutta sen minimipituus on 2 tavua ja maksimipituus 19 tavua[15]. Käytännössä yhteisen otsakkeen sisältö vaihtelee hieman riippuen siitä kumman tyyppinen paketti on kyseessä, julkinen vai yksityinen QUIC-paketti. Otsake muodostuu aina julkisista lipuista(eng. public flags) ja pakettinumerosta. Lisäksi mahdollisia osia ovat QUIC-versionumero sekä yhteys-id.



Kuva 5.1: QUIC-otsake

Kuvassa 5.1 näkyvä ensimmäinen tavu QUIC-otsakkeessa käytetään yleisten asetusten määrittämiseen (Public Flags). Tavun ensimmäinen bitti on varattu QUIC-session versioneuvotteluja varten. QUIC-yhteys alkaa aina asiakkaan ja palvelimen välisellä versioneuvottelulla, jossa molemmat osapuolet hyväksyvät käytettävän version. Toisella bitillä ilmaistaan, että kyseessä on Public Reset -paketti. Kolmas ja neljäs bitti on varattu yhteys-id:n koon ilmaisua varten. Asiakkas ja palvelin voivat neuvotella yhteys-id:n koon. Esimerkiksi asiakas voi pyytää palvelinta ilmoittamaan yhteys-id:n lyhyempänä. Yhteys-id:n pituus on vaihtuvamittainen ja se voi olla 0-64 tavua. Viides ja kuudes bitti on varattu Frame pakettien käyttöön. Näillä biteillä käytännössä ilmoitetaan paketin pakettinumeron pituus. Niissä paketeissa, joissa ei käytetä pakettinumeroa on näiden bittien oltava asetettuna nollaksi. Varatun tavun viimeiset kaksi bittiä on toistaiseksi käyttämättömiä, mutta ne on varattu tulevaisuuden tarpeita varten. Toistaiseksi nämä bitit tulee aina asettaa nollaksi.[14]

Yhteys-id:een kentän pituus vaihtelee nollasta bitistä kahdeksaan tavuun. Yhteys-id:een valinta tapahtuu ensisijaisesti asiakkaan toimesta. Yhteys-id:n tarkoituksena on uniikisti tunnistaa asiakas, riippumatta asiakkaan muista tunnusmerkeistä, kuten IP-osoitteesta, lähtöportista(eng. sourceport) tai kohdeportista(eng. destination port).

QUIC-versionumerolle on varattu maksimissaan neljä tavua pitkä kenttä. Versionumerokenttää käytetään ainoastaan, jos yleisissä asetuksissa on ensimmäinen bitti päällä ilmaisemassa, että versioneuvottelu on käynnissä. Käytännössä versionumero kenttää käytetään niin, että asiakas ilmoittaa

kentällä palvelimelle version, joka asiakkaalla on kyseisen viestin lähetyksessä käytössä. Palvelin käyttää versionumerokenttää vain silloin, jos asiakkaan ilmoittama versionumero ei ole tuettu palvelimella. Ilmoitus tuettomasta versiosta tapahtuu niin, että palvelin vastaa asiakkaan versioehdotukseen listalla tuetuista versioista. Palvelimen ilmiotettua asiakkaan on valittava yksi tuettu versio, jonka jälkeen yhteydessä käytetään vain valittua versiota protokollasta.[14]

Pakettinumerokenttä on joko yhden, kaksi, neljä tai kuusi tavua pitkä. Kentän pituus riippuu siitä, kuinka pitkäksi se on ilmoitettu yleisissä asetuksissa. Pakettinumero on juokseva luku, joka alkaa yhdestä yhteyden alkaessa. Lähettävä osapuoli aloittaa laskennan ja jokainen seuraava paketti sisältää pakettinumeron, joka on aina yhden numeron suurempi kuin edeltäjänsä. Jos yhteys saavuttaa pakettinumeroiden maksimin, joka on 2^{64-1}, täytyy viimeisen viestin lähettäjän ilmaista yhteyden katkaisusta. Tämän jälkeen voidaan aloittaa uusi yhteys uudella pakettinumerolaskurilla. Pakettinumeron avulla QUIC myös rajoittaa lähetyksessä olevien pakettien määrää. QUIC määrittää, ettei lähettäjä voi lähettää kuin 2^{46} pakettia ilman, että saa vastapuolelta ilmoituksen pakettien saapumisesta. Tarkoituksena tällä rajoittamisella on se, ettei vahingossa muodostu tilannetta, jossa pakettinumeron maksimiarvo ylitetään siitä syystä, että matkalla olevat paketit eivät ole vielä saapuneet vastaanottajalle. [15]

5.5 Ominaisuudet HTTP/2-protokollalle

QUIC-protokolla on alunperin suunniteltu toimimaan yhdessä Googlen kehittämän SPDY-protokollan kanssa, josta sittemmin tuli HTTP/2-protokollan runko. QUIC-protokolla sisältää useita toimintoja, jotka on sunniteltu toimimaan nimenomaan HTTP/2-protokollan kanssa yhdessä. Osittain myös QUIC-protokollan toiminnallisuuksia voidaan käyttää korvaamaan kokonaan jokin vastaava toiminnallisuus HTTP/2-protokollasta, ja siten poistaa kompleksisuuttaa HTTP/2-protokollan käytännön toteutuksista. Tässä luvussa tullaan tarkastelemaan QUIC-protokollan HTTP/2-protokollalle tarkoitettuja ominaisuuksia.

Yksi merkittävimmistä ominaisuuksista, jolla QUIC korvaa HTTP/2-protokollan toimintoja on viestivirranhallinta. QUIC-protokolla hallinnoi viestivirtaa HTTP/2-protokollan puolesta silloin, kun HTTP/2 käyttää siirtoprotokollanaan QUIC-protokollaa. QUIC-protokollan viestivirta-id:eistä tulee HTTP/2 -protokollan käyttämiä viestivirta-id:eitä suoraan. Tämä siis

tarkoittaa sitä, että HTTP/2 -protokollan ei tarvitse erikseen hallinnoida päällekkäisiä viestivirta-id:eitä vaan se suoraan omaksuu QUIC:in käyttämät viestivirta-id:eet omikseen. Käytännössä siis QUIC-protokolla poistaa kokonaan HTTP/2-protokollan tarpeen halinnoida viestivirtaa. HTTP/2-protokollan otsakkeet ja tietokentät menevät sellaisenaan QUIC-protokollan siirtämäksi tiedoksi. Tämä toiminto yksinkertaistaa HTTP/2-protokollan toimintaa merkittävästi. Lisäksi tällä QUIC:in toiminnallisuudella poistetaan päällekkäisiä viestivirtoja ja päällekkäistä viestivirranhallinnointia. Ilman QUIC-protokollaa viestivirtaa hallinnoisi sekä HTTP/2 että TCP-protokolla, joissa molemmissa on oma tapansa hallinnoida kulkevaa liikennettä. QUIC korvaa HTTP/2-protokollan hallinnoinnin omallaan, ja näin ollen saa haltuunsa täysivaltaisen kyvyn hallinnoida viestivirtaa. TCP:n toimintamallissa tälläinen ei ole mahdollista, vaan TCP pysyy tiukasta omassa lokerossaa protokollapinossa. Tämä ominaisuus osoittaa erinomaisen hyvin sen, kuinka nykyaikaisten protokollien rinnakkain suunnittelu voi parantaa kokonaisuuden suorituskykyä ja karsia ylimääräistä toistoa rakenteista. [15]

Lisäksi QUIC tarjoaa muita toiminnallisuuksia, jotka on suunniteltu toimimaan nimenomaan yhdessä HTTP/2-protokollan kanssa. QUIC on suunniteltu tarjoamaan täysin pysyviä yhteyksiä asiakkaan ja palvelimen välille, toisin kuin esimerkiksi TCP-protokolla. QUIC luo vain yhden yhteyden palvelimen ja asiakkaan välille, jonka yli onnistuu kaikki osapuolten välinen kommunikointi. Tämä ominaisuus poistaa tarpeen HTTP-protokollan omalle yhteystarkistukselle, eli Connection-kentälle. Connection-kenttä on tarpeellinen siinä tapauksessa, että WWW-istunto jakautuu siirtoprotokollan tasolla useaan yhteyteen. Connection-kentän avulla HTTP-protokolla välittää asiakkaan ja palvelimen välillä tiedon siitä, että kyseessä on kuitenkin vain yksi käytössä oleva WWW-istunto. QUIC-protokollan tarjoama pysyvä yhteys poistaa tarpeen Connection-kentän käytöstä ja näin ollen keventään osaltaan HTTP-otsakkeiden kokoa.

QUIC:in kehitysryhmä pyrkii myös kehittämään QUIC-protokollaa niin, että se soveltuu paremmin HTTP/2-protokollan käyttämän HPACK-pakkauksen kanssa. Tällä hetkellä HPACK tuottaa QUIC-protokollassa tarpeetonta viivettä, sillä pakattujen HTTP-otsakkeiden käsittely jakautuu QUIC-protokollassa pahimmassa tapauksessa useaan viestivirtapakettiin. Usean viestivirtapaketin käsittely kokonaisuutena riippuu jokaisen pakatun otsakkeen käsittelyajasta, mikä tarpeettomasti aiheuttaa viivettä rinnakkaisiin toimintoihin. HTTP-protokolla myös tarjoaa QUIC-protokollaa varten Alternate-Protocol-kentän. Alternate-Protokol-kentän avulla siirtymävaiheessa QUIC-protokolla voi neuvotella sujuvasti HTTP-protokollan sisällä asiakkaan kanssa QUIC:in käytöstä TCP-protokollan sijaan. Tämän neuvottelun on tarkoi-

tus tarjota asiakkaalle helppo rajapinta kokeilla QUIC-protokollaa, kuitenkin niin että palaaminen toimivaan TCP-protokollaan onnistuu myös luontevasti ilman merkittävään virhetilaan joutumista. [15]

Luku 6

QUIC:in ja HTTP/2.0 edut ja mahdollisuudet

QUIC ja HTTP/2.0 ovat molemmat vuonna 2015 esiteltyjä protokollia. Protokollat on suunniteltu nimenomaan korvaamaan ja uudistamaan totaalisesti aiemmin vastaavia toimintoja tarjonneet protokollat. Protokollat parantavat selkeitä virheitä ja vikoja aiemmissa protokollissa. Lisäksi protokollat tarjoaa synergiansa ansiosta merkittäviä uusia mahdollisuuksia. Erityisesti QUIC-protokolla on suunniteltu pitkälti HTTP/2.0 protokollaa varten. QUIC pitää sisällään useita ominaisuuksia, jotka hyödyttävät erityisesti WWW-liikenteessä käytettyä HTTP-protokollaa. Tässä luvussa käsitellään protokolla kerrallaan niiden esiteltyjen uudistusten joukosta niitä ominaisuuksia, jotka saattaisivat avata uusia mahdollisuuksia viiveiden vähentämisen kannalta WWW-liikenteelle.

6.1 HTTP/2.0

HTTP/2 -protokolla on saavuttanut IETF:n standardointi organisaatiossa ”proposed standard”-tilan. Tämä tarkoittaa sitä, että IETF:n on vahvistanut HTTP/2 -protokollan protokollakuvauksen olevan riittävän vakaa yleiseen käyttöön, ja kaikin osin IETF:n tarkistama ja arvioima protokollakuvaus. Käytännössä siis IETF suosittaa HTTP/2 protokollan käyttöönottoa, joka sinällään antaa vahvan indikaation siitä, että ainakin joiltain osin on menty eteenpäin verrattuna aiempaan protokollaversioon.

HTTP/2 -protokollan merkittävimpiä etuja on sen nykyaikaisuus ja kyky vastata tämänpäivän WWW-liikenteen tarpeisiin, kuitenkin ilman että HTTP-

protokollan alkuperäinen toimintamalli uudistuisi kovinkaan merkittävästi. HTTP-pyynnöt ja vastaukset pysyvät lähes täysin muuttumattomina, edelleen GET-pyynnöllä saadaan WWW-sivuston sisältö pyydettyä palvelimelta ja edelleen 404-virhekoodi tarkoittaa, ettei sivustolta löydy pyydettyä resurssia.

Protokollan binäärisyys on yksi tärkeimmistä ominaisuuksista protokollan uudistamisessa. Binäärinen protokolla vaikuttaa suurilta osin WWW-liikenteen viiveiden vähentämiseen. Aiemmin protokolla on käyttänyt tekstienkoodattua protokollamallia, joka ei nykyisessä tietoliikennekentässä ole kovinkaan tehokas. Tekstienkoodatun protokollan purkaminen, pakkaaminen ja siirto on jokaisessa vaiheessa hitaampaa, työläämpää ja hankalampaa verrattuna binääriseen protokollaan. Erityisesti nykyisen WWW-liikenteen kannalta siirrettävän tietoliikennedatan määrä vähenee suhteessa, sillä yhdellä bitillä saadaan siirrettyä sama informaatio johon aiemmin vaadittiin jopa useiden tavujen pituisia tekstikenttiä. Lisäksi protokollaviestien käsittely palvelimen päässä nopeutuu binäärisyyden myötä. Palvelimen toiminnan koodaaminen on merkittävästi helpompaa ja koodin suorittamat operaatiot ovat suoraviivaisempia ja nopeampia kun käsitellään bittejä. Bittien käsittelyssä on käytännössä kaksi vaihtoehtoa, joko bitti on päällä tai bitti ei ole päällä. Binäärisyys lisää protokollan kompleksisuutta jonkin verran, mutta nykyinternetissä binääriset protokollat ovat yleisiä.

HTTP/2 -protokollan esittelemä viestivirta ajattelu HTTP-liikenteessä ratkaisee monta aiempaa ongelmaa HTTP-protokollassa. Viestivirran avulla HTTP-protokolla aidosti hallitsee oman viestiliikenteensä, mikä auttaa protokollaa välttämään tilanteita, joissa kadonnut paketti aiheuttaa viivettä koko viestivirralle. Aiemmin HTTP-protokolla on pitänyt huolen käytännössä vain WWW-sivustoihin ja -pyyntöihin liittyvät kyselyt, ja TCP-protokolla on huolehtinut täydellisesti tiedon siirtämisestä. Ongelmia on aiheutunut WWW-liikenteen pirstaloitumisesta useisiin osiin sivustoissa, joiden hakua varten TCP-protokolla luo oletuksena aina uuden yhteyden. Lisäksi jonkin sivuston osan puuttuminen välistä on saattanut pahimmassa tapauksessa aiheuttaa koko sivuston lataamattomuuden, mikä on asiakkaan kannalta tuskallinen tilanne. HTTP-protokollan tullessa tietoiseksi viestivirran liikenteestä edellä kuvatut ongelmat vähenee, sillä palvelin on kokoajan tietoinen kokonaisuuksista viestivirrassa eikä vain yksittäisistä yhteyksistä. Toisaalta tämä aiheuttaa jonkinlaisen ongelman protokollien välisessä hierarkiassa, sillä käytännössä HTTP-protokolla luo päällekkäisen toiminnon alla toimivan TCP-protokollan ohelle. HTTP/2-protokollassa tämä päällekkäisyys on huomioitu mahdollisuudella käyttää QUIC-protokollan tarjoamaa viestivirranhallintaa, mutta TCP-protokollan kanssa toimiessa vastaavaa mahdollisuut-

ta ei ole mikä aiheuttaa periaatteessa päällekkäisiä toimintoja protokollapinossa. Käytännössä kuitenkin HTTP/2-protokollan viestivirta ei hoida siirtoprotokollan tehtäviä, mutta siirtovuonhallinta toteutetaan kaksinkerroin, sekä HTTP- että TCP-protokollassa. Tälläinen toiminta on vahvasti ristiriidassa protokollapinon perinteisessä työnjaossa. Ilmeisesti kuitenkin jo tässä vaiheessa on jonkinlainen päätös olemassa siitä, että QUIC-protokolla tulee korvaamaan TCP-protokollan WWW-liikenteessä, jonka myötä tämä ristiriita poistuu HTTP-protokollan käyttäessä QUIC:in viestivirranhallintaa. Toisaalta myös TCP-protokollan toiminnan kannalta HTTP/2-protokollan oma viestivirta helpottaa ruuhkanhallintaalgoritmien toimintaa, sillä HTTP/2 kykenee suorittamaan rinnakkaisia operaatioita yhden yhteyden sisällä. Aiemmin TCP:n ruuhkanhallintaa on ollut hankalaa toteuttaa käytännössä, kun HTTP-yhteydet ovat jakaantuneet useaan erilliseen TCP-yhteyteen, joilla jokaisella on itsenäinen ruuhkanhallinta. Kaiken kaikkiaan HTTP/2 -protokolla on merkittävä parannus viestivirranhallintaan ja yhteyden ylläpitoon asiakkaan ja palvelimen välillä.

Verrattuna vanhempaan protokollaan on HTTP/2-protokolla suunniteltu kehittymään ja toimimaan tulevaisuudessa. HTTP/2-protokollassa esitellään selkeästi, kuinka protokolla on laajennettavissa. Protokollan suunnittelu on ollut toki helpompaa nykypäivänä, sillä internet on vakiintunut osa yhteiskuntaa ja sen tarve yhteiskunnalle on merkittävästi selkeämpi kuin mitä se on ollut vuonna 1997. HTTP/2 voi joiltain osin olla siinä asemassa, että sen ominaisuuksia käytetään hyödyksi IoT-laitteiden hallinnassa.

6.2 QUIC

QUIC-protokolla on toistaiseksi vielä luonnos internet-standardiksi. Protokollasta on julkaistu kolme versiota IETF:n ylläpitämässä arkistossa, ensimmäinen versio ja kaksi päivitystä siihen. Protokolla on selkeästi kehittynyt versioiden välillä ja sen kehitystyö on aktiivista, sillä päivityksiä on julkaistu noin neljän kuukauden välein. QUIC-protokollan kehitystyöhön olennaisesti osallistuva Google tarjoaa QUIC-protokollaa käytettäväksi omassa selaimessaan ja omissa palveluissaan. Protokolla on siis käytännössä internetin käyttäjien testauksessa jo ennen kuin se on suositeltu standardi. Google saakin merkittävää hyötyä siitä, että se pääsee testaamaan protokollan toimintaa käytännössä ja pystyy kerämään käyttöädataa sekä palvelimen että asiakkaan päässä toimivuuteen.

QUIC vaikuttaa tekevänsä vakaavasti tuloaan internetin siirtoprotokollak-

si. Se tuo tullessaan merkittäviä etuja ja onnistuu suoriutumaan WWW-liikenteen siirrosta nopeammin kuin nykyisin yleisesti käytössä oleva TCP-protokolla[5]. Lisäksi QUIC:in ehdottomia vahvuuksia verrattuna TCP-protokollaan on sen HTTP/2-protokollan kanssa rinnakkain suunnitellut toiminnallisuudet.

Merkittävä helpotus kaikelle verkkoliikenteelle olisi QUIC:in mukanaan tuoma yhteyksien vähennys. TCP-protokollan keskeisimpiä ongelmia on sen useiden rinnakkaista yhteyksien luonti, joka taas johtuu nykyisten WWW-sivustojen rakenteesta. Sivustojen rakenne tuskin muuttuu merkittävästi, sen sijaan siirtoprotokollalle on jo nyt tarjolla vaihtoehto. Useat rinnakkaiset yhteydet ovat ongelmallisia sekä asiakkaalle, palvelimelle, että välissä toimiville verkkolaitteille. Rinnakkaisten yhteyksien hallinnointi yhtenä kokonaisuutena ei nykyisillä tekniikoilla onnistu. Pahimmassa tapauksessa yhden asiakkaan luomat rinnakkaiset yhteydet kilpailevat siirtokaistasta keskenään. Tämä saa aikaan pahimmassa tapauksessa sen, että asiakas ei saa kelvollista palvelua palvelimelta ja verkolta, koska se itse tukahduttaa olemassa olevat resurssit. TCP on pyrkinyt vastaamaan tähän ongelmaa useilla eri tavoilla, esimerkiksi eri tyyppisillä jononhallinta algoritmeillä, mutta käytännössä kuitenkin ongelma tulee TCP:n rakenteesta, minkä takia sen korjaaminen vastaamaan nykyistä verkkoliikennettä on erittäin hankalaa. QUIC vastaa nimenomaan tähän ongelmaan tarjoamalla aidosti yhden yhteyden, jonka sisään kootaan kaikki asiakkaan ja palvelimen välillä tarvittavat HTTP-yhteydet. Yhden yhteyden tapauksessa jononhallinta, vuonhallinta ja kokonaisuuden käsittely on merkittävästi yksinkertaisempaa, kun kaikki tieto on olemassa yhteyttä hallinnoivalla protokollalla. QUIC:n ei aidosti tarvitse ottaa huomioon yhteyksiä saman arvoisina, vaan se voi rehellisesti priorisoida liikennettä asiakkaan ja palvelimen välillä.

QUIC:in etuihin sisältyy myös sen lyhyempi kättely yhteyden avauksessa. TCP:n historiallisista syistä oleva kolmitiekättely on nykyisessä verkkoliikenteessä osittain tarpeeton, mutta ennen kaikkea auttamattoman hidas. Varsinkin yleinen verkkoinfran kehittyminen ja verkkopalveluntarjojien palvelutasonnosto tekee kolmitiekättelystä pullonkaulan verkkoyhteyden viiveille. Asiakkaalta aina palvelimelle saakka yhteyksien laatu on parantunut TCP:n suunnitteluajoista, eikä enää ole olemassa yhtä suurta riskiä siitä, että asiakas epäonnistuu yhteyden muodostamisessa jos alkumetreillä. Oletuksena tänä päivänä on, että yhteys saadaan muodostettua eikä se lähtökohtaisesti jää kiinni verkon toimimattomuudesta. QUIC lupaa parhaimmillaan suoriutuvan kättelystä käytännössä ilman kättelyä, ja heikoimmillaankin vähintään yhtä nopeasti kuin TCP:kin. Kättelyn nopeutuminen tulee vaikuttamaan merkittävästi koko yhteyden viiveisiin. Kerrannaisena yksittäisen kättelyn no-

peutumiselle toimii vielä QUIC:in parempi yhteyksien hallinta, jolloin kättelyjä tulee vähemmän. Lukumäärällisesti vähemmän kättelyjä ja nopeampi kättelyistä suoriutuminen ovat ominaisuuksia, jotka ainakin teoriassa tulisivat laskemaan hyvinkin merkittävästi koko WWW-liikenteelle syntyvää viivettä. Yhteyksien kapasiteetti keskittyisi paremmin ja tehokkaammin varsinaiseeen datan siirtoon.

Monia QUIC-protokollan ominaisuuksia on esitetty aiemmin myös TCP-protokollaan liitettäväksi. Esimerkiksi Google esitteli SIGCOMM-tapahtumassa useita ominaisuuksia, jotka käytännössä on toteutettu muutama vuosi myöhemmin QUIC-protokollaan.[12] Kuitenkin vastaavien ominaisuuksien kehittäminen ja mahdollistaminen TCP-protokollassa ei ole mahdotonta, ja sen eteen Googlen tekemän työ on jo jonkinlainen alku. Tämä entisestään kuroo kiinni QUIC:in etumatkaa TCP-protokollaan nähden. Esimerkiksi FEC-toiminnallisuuden olemassa olo QUIC-protokollassa on erinomainen ominaisuus, mutta käytännössä jos se on mahdollista toteuttaa myös TCP-protokollassa pienellä vaivalla voi olla, että hyöty QUIC-protokollaan siirtymisellä jää pieneksi. Toki QUIC-protokolla on alusta asti suunniteltu niin, että se käyttää FEC-ominaisuutta, joka helpottaa monilta osin ominaisuuden käytännön implementointia. Jää nähtäväksi, kuinka TCP-protokollan kehitys vaikuttaa QUIC-protokollaan, ja kuinka QUIC-protokollan kehitys vaikuttaa Googlen haluun kehittää muita protokollia, kuten esimerkiksi TCP-protokollaa.

QUIC:in suorityskykyä on tutkittu ja verrattu TCP:hen sekä HTTP/1.1 että HTTP/2 -yhteyksillä. Käytännössä kuitenkin QUIC:in ollessa vielä voimakkaasti kehityksen alainen, on sen testaaminen lähinnä suuntaa antavaa, sillä Google myöntää avoimesti vielä QUIC:ssä olevia ongelmakohtia. Ongelmakohdista kertoo sekin, että uusimmasta protokollan vedoksesta löytyy neljä kohtaa merkattuna merkinnällä ”TODO”, mikä tarkoittaa, että tehtävää vielä löytyy.

Esimerkiksi Gaetano Carluccin työryhmän tekemä tutkimus [5] antaa jonkinlaisen käsityksen siitä mihin QUIC:n avulla voitaisiin päästä. Tutkimus on julkaistu vuonna 2015, joten kyseessä ei ole ainakaan viimeisin QUIC:in versio. Työryhmä esittää, että QUIC:in virheitä korjaava FEC-ominaisuus aiheuttaa merkittävää haittaa protokollan kokonaissuorituskyvylle. Kuitenkin merkittävä tunnustus QUIC:lle tulee siitä, että se pienentää WWW-sivuston kokonaislatausaikaa verrattuna testissä olleeseen TCP-verrokkiin. Erityisesti QUIC:in nopeus tulee esille suuria sivustoja ladatessa, jolloin haettavien sivustonosien määrä kasvaa. Huomionarvoista on se, että kyseessä on tietty TCP-algoritmi. Algoritmi on valittu testiin yleisyytensä vuoksi, joten ky-

seessä on hyvä yleiskatsaus QUIC:in suhteesta WWW-liikenteeseen. Tutkimus kuitenkin osoittaa sen, ettei QUIC ole vielä ainakaan täysin ylivertainen suhteessa TCP:hen vaan, että TCP:lläkin löytyy ehdottomia vahvuuksia, jotka ainakin toistaiseksi toimivat paremmin kuin QUIC:in vastaavat. Esimerkiksi verkoissa, joissa on paljon pakettihäviötä, on TCP edelleen hyvin vahva ja suorituskykyinen kuten tähänkin asti. QUIC-protokolla tarjoaa loistavan mahdollisuuden tuoda WWW-liikenne kokonaisuudessaan UDP:n päällä toimivaksi. QUIC ei ole vielä valmis protokolla, mutta sen toimintatavat ovat loistava mahdollisuus verkkoliikenteen uudistamiselle. Tulevaisuus näyttää, kuinka QUIC kehittyy ja tuleeko se saavuttamaan tai syrjäyttämään TCP:n WWW-liikenteen kuljetusprotokollana.

Luku 7

Yhteenveto ja johtopäätökset

Tässä työssä on esitelty WWW-liikenteen nykytila ja tutkittu sen mahdollista tulevaisuutta. Työssä on esitelty nykyisin yleisessä käytössä olevat TCP ja HTTP -protokollat, ja niiden ongelmat nykyisen WWW-liikenteen kannalta. Työn tarkoituksena on ollut kartoittaa tällä hetkellä tarjolla olevia vaihtoehtoja nykyisten WWW-liikenteessä käytettyjen protokollien tilalle ja esitellä uusia tekniikoita, joilla voitaisiin vähentää viiveitä nykyisessä WWW-liikenteessä. Työssä on keskitytty erityisesti uusien protokollien, HTTP/2 ja QUIC, esittelyihin. Esitellyt protokollat pyrkivät erityisesti paikkaamaan edeltävien protokollien havaittuja rakenteellisia ongelmia, joita on käyty läpi työn 2. ja 4. luvuissa. Molemmat protokollat HTTP/2 ja QUIC keskittyvät vahvasti viiveiden minimointiin WWW-liikenteessä.

HTTP/2-protokolla on hyväksytty internetstandardiksi keväällä 2015. Protokolla uudistaa HTTP-protokollaa monilla tavoin, esimerkiksi muuttamalla protokollan binääriseksi ja esittelemällä viestivirrat. Käytännössä HTTP/2-protokolla ei vielä kuitenkaan ole korvannut edeltäjäänsä täysin. HTTP/2-protokolla on lupaava protokollakuvaus, jonka uudistukset HTTP-protokollaan ovat kaivattuja ja tarpeellisia. HTTP/2-protokollan käyttö on yleistynyt merkittävästi työn kirjoituksen aikana, ja vaikuttaisi siltä että HTTP/2 on vauhdilla korvaamassa edeltäjäänsä. HTTP/2-protokollan tuki löytyy tällä hetkellä kaikista käytetyimmistä verkkoselaimista. Lisäksi tuki HTTP/2-protokollalle on lisätty useimpiin HTTP-palvelinohjelmistoihin. HTTP/2-protokollan tuomat uudistukset ovat selvästi olleet kaivattuja. Uudistusten toteuttaminen on onnistuttu tekemään hyvin harkiten ja lopputuloksesta on tullut tasapainoinen ja toimiva korvaaja aiemmalle protokollaversiolle. Lähivuosina HTTP/2-protokolla tulee varmasti korvaamaan lähes täydellisesti nykyisin WWW-liikenteessä käytettävän HTTP/1.1-protokollan.

Lisäksi työssä on käyty läpi TCP:n korvaajaksi suunniteltu kuljetuskerroksen ja sovelluskerroksen välillä toimiva QUIC-protokolla. QUIC-protokollaa kehittää Googlen työryhmä, joka on toistaiseksi päivittänyt protokollakuvausta kolmen-neljän kuukauden välein. QUIC-protokolla on vahvasti optimoitu HTTP-liikenteelle. Se sisältää erityisesti HTTP-protokollaa varten kehitettyjä ominaisuuksia, kuten esimerkiksi mahdollisuus HTTP-protokollan viestivirranhallinnan korvaamiseen QUIC:in viestivirranhallinnalla. QUIC-protokolla ei periaatteessa ota kantaa sen kuljettamaan tietoon, mutta käytännössä optimointi HTTP-protokollalle on viety monilta osin niin pitkälle, että se sinällään yksipuolistaa QUIC-protokollan mahdollisuuksia. TCP-protokolla ainakin nykyisellään tuntuisi olevan merkittävästi kattavampi ratkaisu kuljetuskerroksen protokollaksi verkkoliikenteessä, kuin QUIC-protokolla. QUIC-protokollassa on useita parannuksia kuljetuskerrokselle, mutta käytännössä niiden toimivuuden osoittaminen ainakin toistaiseksi on jäänyt jonkin verran kyseenalaiseksi. Riippumattomia QUIC-protollan käytännön testituloksia on toistaiseksi julkaistu hyvin vähän. Testaaminen jättää varaa tulevaisuuden tutkimuksille. Protokollan suorituskyvyn osoittaminen erilaisissa skenaarioissa HTTP/1.1, HTTP/2 -protokollien kanssa tulee osoittamaan QUIC-protokollan todellisen suorituskyvyn nykyisessä WWW-liikenteessä. Esimerkiksi langattomista ympäristöistä ei vielä tämän työn kirjoittamisen aikaan ollut tarjolla ainuttakaan tehtyä tutkimusta. QUIC-protokolla saattaa korvata TCP:n WWW-liikenteen kuljetusprotokollana, tosin sekin vaatii vielä jonkin verran kehitystä QUIC-protokollaan. QUIC-protokolla tuskin korvaa TCP:n yleisesti kuljetuskerroksen protokollana. Mahdollisuus sille, että QUIC voisi korvata TCP:n kuljetuskerroksen protokollana vaatisi mielestäni käytännössä sen, että kaikki sovelluskerroksen verkkoliikenne siirtyisi käyttämään HTTP-protokollaa. Googlen näkökulmasta kaiken liikenteen siirtyminen HTTP:n päälle on hyvinkin mahdollinen skenaario, minkä takia on täysin ymmärrettävää, että Google on lähtenyt panostamaan QUIC-protokollan kehitykseen.

TCP-protokolla on pitkäaikainen ja melko hyvin kehittyvä ja sulautuva protokolla, joka on vuosien varrella osoittanut tarpeellista kykyä muuntautua vastaamaan ympäristön muutoksia. TCP ei välttämättä ole paras mahdollinen protokolla kaikelle liikenteelle, mutta se on ehdottomasti paras optimiointi valtaosalle liikenteestä. Lisäksi TCP-protokolla on hyvinkin kiinteästi juurtunut käyttöjärjestelmien ja verkkolaitteiden ytimiin, joten sen korvaaminen, kuten QUIC:in kehittäjätkin tunnustavat on kuljetuskerroksella käytännössä mahdotonta. Nähtäväksi jää, kuinka TCP-protokolla pärjää uudistetun HTTP/2.0-protokollan kanssa. Uudessa sovelluskerroksen protokollassa esitelty viestivirranhallinta saattaa aiheuttaa yllättäviä ongelmia

TCP-protokollan kannalta.

Lähteet

[1] Authors, T. C. The chromium projects, 2016. WWW Google Chromium webbrowser development crew `https://www.chromium.org/quic` `http://blog.chromium.org/2015/04/a-quic-update-on-googles-experimental.html`. Viitattu 22.09.2015.

[2] Belshe, M., Peon, R., and Google Inc. SPDY protocol - draft 1, 2009. WWW blog hosted by Google Chromium webbrowser development crew `https://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft1`. Viitattu 18.01.2015.

[3] Belshe, M., Twist, Peon, R., Google Inc., Thomson, M., Microsoft, and Melnikov, A. SPDY Protocol - draft-ietf-httpbis-http2-00. Tech. rep., Isode Ltd, 2013.

[4] Berners-Lee, T., Fielding, R., and Frystyk, H. Hypertext transfer protocol–HTTP/1.0. Tech. rep., IETF, 1996.

[5] Carlucci, G., De Cicco, L., and Mascolo, S. HTTP over UDP: An experimental investigation of QUIC. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing* (New York, NY, USA, 2015), SAC '15, ACM, pp. 609–614.

[6] Chu, H. J., Dukkipati, N., Cheng, Y., and Mathis, M. Rfc6928 - increasing tcp's initial window. Tech. rep., IETF, 2013.

[7] Chu, J., Jain, A., Cheng, Y., and Radhakrishnan, S. Tcp fast open. Tech. rep., IETF, 2014.

[8] Fall, K., and Stevens, W. *TCP/IP illustrated, volume 1: The protocols*. addison-Wesley, 2011. Second Edition.

[9] Fielding, R., Adobe, Reshke, J., and greenbytes. Hypertext transfer protocol (HTTP/1.1): Message syntax and routing. Tech. rep., IETF, 2014.

[10] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. Hypertext transfer protocol–HTTP/1.1. Tech. rep., IETF, 1999.

[11] Fielding, R., Irvine, U., Gettys, J., Mogul, J., DEC, Frystyk, H., Berners-Lee, T., and MIT/LCS. Hypertext transfer protocol – HTTP/1.1. Tech. rep., IETF, 1997.

[12] Flach, T., Dukkipati, N., Terzis, A., Raghavan, B., Cardwell, N., Cheng, Y., Jain, A., Hao, S., Katz-Bassett, E., and Govindan, R. Reducing web latency: the virtue of gentle aggression. *ACM SIGCOMM Computer Communication Review 43*, 4 (2013), 159–170.

[13] Google Inc. SPDY white-paper. `http://dev.chromium.org/spdy/spdy-whitepaper`. Viitattu: 1.10.2015.

[14] Hamilton, R., Iyengar, J., Swett, I., Wilk, A., and Google Inc. QUIC: A UDP-based secure ja reliable transport for HTTP/2, draft-tsvwg-quic-protocol-01. Tech. rep., IETF, 2015.

[15] Hamilton, R., Iyengar, J., Swett, I., Wilk, A., and Google Inc. QUIC: A UDP-based secure ja reliable transport for HTTP/2, draft-tsvwg-quic-protocol-02. Tech. rep., IETF, 2016.

[16] Henderson, T., Floyd, S., and Gurtov, A. The newreno modification to tcp's fast recovery algorithm. Tech. rep., IETF, 2012.

[17] HTTPWatch. A simple performance comparison of HTTPS, SPDY and HTTP/2. `https://blog.httpwatch.com/2015/01/16/a-simple-performance-comparison-of-https-spdy-and-http2/comment-page-1/`. Viitattu 12.03.2016.

[18] Kuehlewind, M., Hazel, G., Shalunov, S., and Iyengar, J. Low extra delay background transport (ledbat). Tech. rep., IETF, 2012.

[19] Langley, A., and Chang, W.-T. QUIC crypto. Tech. rep., Google, 2014.

[20] Leiba, B., and IETF. Hypertext transfer protocol WG charter. `https://datatracker.ietf.org/doc/charter-ietf-httpbis/`. Viitattu 04.01.2016.

[21] Peon, R., Ruellan, H., Google Inc., and Canon CRF. HPACK: Header compression for HTTP/2. Tech. rep., IETF, 2015.

[22] Postel, J. RFC 793: Transmission control protocol, september 1981. Tech. rep., IETF, 1981.

[23] Tarreau, W., Exceliance, Jeffries, A., Treehouse Networks Ltd., de Croy, A., Qbik New Zealand Ltd., and Kamp, P. Proposal for a network-friendly HTTP upgrade - draft-tarreau-httpbis-network-friendly-00. Tech. rep., Varnish Cache Project, 2012.

[24] Thomson, M. Hypertext transfer protocol version 2 (HTTP/2). Tech. rep., IETF, 2015.

[25] Trace, R., Foresti, A., Singhal, S., Mazahir, O., Nielsen, H., and Montenegro, G. HTTP speed+mobility - draft-montenegro-httpbis-speed-mobility-00. Tech. rep., Microsoft, 2012.