OULUN YLIOPISTO
UNIVERSITY of OULU

Nykypäivän Internet ja selainteknologiat

Oulun Yliopisto
Tietojenkäsittelytieteiden
koulutusohjelma
Kandidaatintyö
Jonne Nauha
17.5.2016

Tiivistelmä

Tietoliikenneverkot ja niissä toimivat protokollat ovat Internetin ydin. Ne muodostavat alla olevan putkiston joka mahdollistaa kaiken, mitä pidämme itsestäänselvyytenä, kun hyödynnämme Internetiä liiketoiminnassa sekä arjen työkaluna. Yleinen ihmisten tekemä trivialisointi, että Internet on sarja taianomaisia putkia, jotka kuljettavat tietoa on osuva kuvaus maallikolle. Viime aikoina pilvipalvelut-termi on vahvistanut ihmisten kuvaa Internet-jumalista pilvien päällä, jotka jakelevat sinulle suosikkisivustosi. Todellisuus on paljon maanläheisempi.

Lähestymme aihealuetta Internetin loppukäyttäjän sekä sovelluskehittäjän näkökulmasta. Käymme läpi Internetin mahdollistavia protokollia, standardeja ja teknologioita, joita nykypäivän selainpohjaiset sovellukset ja Web-sivustot hyödyntävät. Näitä ovat sovelluskerroksen HTTP-, WebSocket-, ja WebRTC-protokollat, sekä JavaScript-, WebGL- ja WebComponents-selainteknologiat.

Sisällysluettelo

Tiivistelmä 2
Sisällysluettelo 3
1. Johdanto 4
2. Verkko- ja kuljetuskerros 5
3. HTTP 8
    3.1. Viestin rakenne 8
    3.2. TCP-yhteydet 9
    3.3. Otsikkokentät 9
    3.4. Salaus 10
    3.5. Protokolla-version neuvottelu 12
    3.6. SPDY ja QUIC 12
    3.7. HTTP/2 kriittinen vastaanotto 13
4. WebSocket 14
    4.1. Ohjelmointirajapinta 14
5. WebRTC 15
6. Selainteknologiat 16
    6.1. JavaScript 16
    6.2. WebGL 16
    6.3. WebComponents 17
7. Yhteenveto 18
Lähteet 19

1. Johdanto

Nykypäivän Internet ja sen tarjoamat sovellukset muuttuvat rivakasti. Web-alustan tarjoamat ohjelmointirajapinnat sekä sen alla operoivat verkkoprotokollat ovat kehittyneet muutaman viime vuoden aikana uuteen suuntaan. Uusien standardien ja Web-selainten selvä päämäärä on päästä lähelle työpöytäsovellusten toiminnallisuutta. Mobiilipäätelaitteet ovat nyt Web-alustalle tärkeä kohde. Monet maailman käytetyimmistä sivustoista vastaanottavat enemmän liikennettä mobiililaitteista kuin perinteisistä työpöytäympäristöistä. Yritykset haluavat tarjota näille käyttäjille mahdollisimman nopean ja kattavan palvelun.

Nykypäivän Web-sovellukset voivat lähettää ilmoituksia puhelimeesi ilman, että sivu on enää auki. Sovellukset voivat tarjota sisältöä ilman Internet-yhteyttä. Käyttämällä uusimpia protokollia ja teknologioita sovellukset latautuvat nopeammin. Reaaliaika-yhteyksiä voidaan käyttää sovelluksen ajonaikaiseen päivitykseen. Video- ja äänipuheluiden tekeminen on helppoa ilman selainlisäosia. HTML5 ja muut standardit jättävät antiikkiset Flash-sovellukset ja muut lisäosat historiaan.

Tutkin nykypäivän isoimpia muutoksia sovelluskerroksen sekä sovelluskehityksen näkökulmasta. Tulevaisuus Web-alustalle näyttää valoisalta. Standardisoinnin ja selainvalmistajien nopeamman kehitystahdin myötä Web-alusta ei ole enää niin pirstoutunut, kuin mitä se oli muutama vuosi sitten. Kova kilpailu selainmarkkinoilla on loppukäyttäjille pelkästään positiivinen asia. Yleinen asenne vanhojen selainten tukemiselle on terveempää. Näen asennoitumista missä uudet sekä vanhat projektit päättävät olla tukematta antiikkisia selaimia, missä standardien toteutus on heikkoa tai olematonta. Kuuluisana esimerkkeinä jokaisen Web-ohjelmistokehittäjän pelko pakollisesta Microsoft Internet Explorer 6-9 -versioiden tukemisesta. Niin sanottuja aina ajantasaisia Web-selaimia, jotka päivittävät itsensä automaattisesti tasaisin väliajoin, on saatavilla jokaiselle käyttöjärjestelmälle ja päätelaitteelle.

Tutkielman rakenne on seuraava: Ensimmäisellä luvussa esitellään korkealla tasolla verkko- ja kuljetuskerroksen tehtävät ja toiminta. Ne muodostavat pohjan sovelluskerroksen protokollille. HTTP-luvussa käydään läpi HTTP-verkkoprotokollaa ja sen eri versioita, jotka ovat Internet-liikenteen kivijalka. Ne vastaavat lähes kaikesta Web-selainten verkkoliikenteestä. WebSocket-luvussa käsitellään Web-alustan uutta reaaliaikaista viestintää, joka on nykypäivänä mahdollista myös palvelimelta asiakasohjelmaan päin. WebRTC-luku esittelee Web-alustan ääni- ja video-ominaisuuksia, jotka mahdollistavat esimerkiksi videopuhelut. Luku kuusi käy läpi nykypäivän selainteknologioita, jotka ovat sovelluskehittäjien käytettävissä.

2. Verkko- ja kuljetuskerros

Verkkokerros (network layer) on vastuussa päätelaitteiden välisestä kommunikaatiosta. Kuljetuskerros luottaa tähän palveluun tietämättä sen toteutuksesta. Korkealla tasolla verkkokerroksessa voidaan tunnistaa kaksi tärkeää tehtävää: reitittäminen (routing) ja uudelleenohjaus (forwarding). Reitittämistä tapahtuu verkon linkeissä tarpeen mukaan, kun paketit etenevät lähettäjältä vastaanottajalle. Ohjausta tapahtuu pääasiallisesti reitittimissä, jolloin ne ohjaavat paketteja seuraavalle reitittimelle. Jokaisessa reitittimessä on uudelleenohjaustaulu, jota indeksoidaan pakettien kehyksestä löytyvällä arvolla. (Kurose & Ross, 2013)

Matalalla tasolla verkkokerroksen ydin on Internet Protocol (IP). Tällä tasolla puhutaan pakettien rakenteesta: otsikosta (header) ja sen kuljettamasta sovelluskohtaisesta datasta. Jokaista Internetissä olevaa laitetta yhdistää se, että ne toteuttavat Internet Protocol -kerroksen.

Jokainen IP-viesti sisältää tarvittavat tiedot tehokkaaseen ja onnistuneeseen reitittämiseen lähettäjältä vastaanottajalle. Viestin kokonaispituuden ilmaisemiseen on varattu 16-bittiä, jolloin paketin teoreettinen maksimipituus on 65635 tavua. Käytännössä paketin koko ylittää harvoin 1500 tavua. (Kurose & Ross, 2013)

Viestin lähde- ja kohdeosoite ovat IPv4:ssä 32-bitin ja IPv6:ssa 128-bitin pituisia kenttiä. Molemmat kentät täytetään lähetyspäässä. Useimmiten kohdeosoite tarkennetaan ja selvitetään ihmisluettavasta palvelinosoitteesta IP-osoitteeksi DNS-palvelujen avulla (Kurose & Ross, 2013).

Internet Protocol versio 6 on ollut kehitteillä 1990-luvulta asti. Tällöin Internet Engineering Task Force (IETF) totesi 32-bittisen IPv4 osoiteavaruuden loppuvan tulevaisuudessa päätelaitteiden määrän kasvaessa räjähtävästi vuosi vuodelta. Realiteetti on kuitenkin, että IPv4 osoiteavaruus loppui virallisesti 3. päivä helmikuuta 2011 (IPv6, 2016). Tällöin Number Resource Organization (NRO) jakoi viimeiset osoitteet alarekistereilleen (Regional Internet Registry, RIR). Nämä alaorganisaatiot osoittavat IP:itä maailman eri Internet-toimijoille niin pitkään, kuin niitä on jäljellä (IPv6, 2016).

Verkkokerroksen hoitaessa kahden päätelaitteen välisen kommunikaation, sitä ylempi kuljetuskerros (transport layer) vastaa kahden sovelluksen välisestä kommunikaatiosta. Kuljetuskerros on alin taso, joka on tietoinen päätelaitteiden sovelluksista. Se vastaanottaa sovelluksen viestit ja segmentoi ne mahdollisesti pienemmiksi paketeiksi sekä lisää protokollan mukaiset otsikkokentät datan eteen. (Kurose & Ross, 2013)

On tärkeää huomioida, että Internet Protocol tarjoaa viestintää kahden päätelaitteen välillä. Sen palvelumalli pyrkii parhaaseen yritykseen, jolloin se ei lupaa datan yhtenäisyyttä, koskemattomuutta tai edes sen perille toimitusta. Tämän takia IP:tä kutsutaan epäluotettavaksi palveluksi (Kurose & Ross, 2013). Jos luotettavuus ja datan yhtenäisyys ovat tärkeitä sovelluskehittäjälle, hänen täytyy valita kuljetuskerros, joka takaa tarvittavat palveluominaisuudet.

TCP/IP verkossa kommunikaation pääpaino on kahdessa kuljetuskerroksen protokollassa. User Datagram Protocol (UDP) joka tarjoaa applikaatioille epäluotettavaa yhteydetöntä viestintää, kun taas Transmission Control Protocol (TCP) tarjoaa luotettavaa viestintää. UDP:n ja TCP:n perimmäisin tarkoitus on jatkaa IP-palvelumallia tarjoten sen päälle uusia takuita. (Kurose & Ross, 2013)

Jotta kuljetuskerros tietää mihin prosessiin kukin viesti ohjataan, se käyttää pistokkeita (socket), jotka ovat aina sidottuja tiettyyn porttiin. Yhteistä kaikille kuljetuskerroksen viesteille on lähettäjän ja vastaanottajan porttinumero. Tämän lisäksi UDP ja TCP kirjoittavat niiden palvelumallille tarpeellisia otsikkokenttiä.

Tietojenkäsittelyssä portit ovat 16-bittisiä lukuja, joka tarkoittaa arvojoukkoa 0-65535. Arvot 0-1023 ovat niin sanottuja tunnettuja portteja, jotka ovat teoriassa varattu tietyille applikaatioille (Kurose & Ross, 2013). Esimerkiksi käytettäessä porttia 80 oletetaan viestien olevan HTTP-liikennettä. Käytännössä kuitenkin mikä tahansa sovellus voi käyttää näitä portteja mihin tarkoitukseen vain. Sovelluskehittäjän täytyy kuitenkin tehdä tämä valinta tiedostaen Internet-infrastruktuurin vaikutukset, kun käyttää porttia johonkin muuhun kuin sen tunnettuun käyttötarkoitukseen. On tunnetusti myös applikaatioita, jotka hyväksikäyttävät esimerkiksi portin 80 ja 8080 ominaisuutta läpäistä helposti tiukimmatkin palomuurit. Käytännön kokemuksesta tiedän, että Skype ja Spotify tekevät näin, jos portit ovat käytettävissä.

UDP:n epäluotettavuutta voidaan yleisesti pitää viestinnässä heikkoutena, mutta tietyt sovellukset hyödyntävät ennen kaikkea sen nopeutta. Käymme seuraavaksi läpi sen vahvuuksia TCP:hen verrattuna. (Anttila, 2000)

UDP lähettää sille annetun datan välittömästi. Se ei toteuta ruuhkautumista hallitsevia mekanismeja tai uudelleenlähetystä kuten TCP. Sovellus voi pilkkoa viestien datan sille optimaalisella tavalla. (Anttila, 2000)

Yhteydetön viestin välitys nopeuttaa viestintää. Se lähettää suoraa datan vastaanottajalle, kuunteli sovellus siellä porttia tai ei.

Kiinteät kustannukset yhteyksille ovat pieniä, koska niiden tilaa ei säilytetä muistissa. TCP ylläpitää tilaa jokaista yhteyttä kohti. Se indeksoi paketit ja viestit sen uudelleenlähetyksen ja muiden mekanismien takia. UDP ei joudu ylläpitämään mitään tilaa ja pakettien otsikkokenttä on vain 8 tavua kun TCP:llä se on 20 tavua. (Anttila, 2000)

UDP soveltuu hyvin reaaliaikajärjestelmiin, joissa luotettavuus voidaan uhrata nopeuden sijasta. Esimerkiksi video- ja äänipuhelut eivät välitä, jos jokin paketti ei saavu perille koskaan. Jos pakettien hävikin määrä (packet loss) on korkea, ääneen tai videokuvaan voi tulla havaittavia artefakteja kuten äänen särinää.

Peliteollisuuden verkkopelit ovat yksi UDP suurkuluttajista. Verkkomoninpeleissä hyödynnetään usein yhtäaikaisesti UDP ja TCP-yhteyksiä. UDP:tä käytetään esimerkiksi objektien liikepakettien lähetykseen. Liikepaketteja lähetetään tiuhaan, tavanomaisesti 10-30 sekunnissa. Kuten video- ja äänipuheluissa muutaman liikepaketin häviäminen ei haittaa sovelluksen toimintaa, koska uusia liikepaketteja on jo tulossa. Pelit voivat käyttävät rinnakkaista TCP-yhteyttä tärkeä datan lähetykseen, kuten pelitilan muutokset, jotka täytyy luotettavasti saada jokaiselle pelaajalle. (Pack, Hong, Park, Kim, Ko, 2002)

TCP on UDP:n luotettava aisapari. Se tarjoaa virheiden havainnointia, kadotettujen pakettien uudelleenlähetyksen sekä pakettien perillemenon varmennusmekanismeja.

TCP on yhteysorientoitunut. Toisin kuin UDP, se tekee yhteyden alussa kättelyn, joka varmentaa, että yhteys on avattu oikeaoppisesti ja että molemmat sovellukset ovat valmiita keskustelemaan. Avaus- ja kättelymekanismit ovat esimerkiksi HTTP liikenteessä merkittävä hidaste (Anttila, 2000). TCP-yhteydellä on paljon tilamuuttujia kuljetuskerroksessa näitä sisäänrakennettuja mekanismeja varten. Sovelluksen ei tarvitse

nähdä eikä välittää tästä tilasta. Verkkokerroksen linkit ja reitittimet eivät pidä TCP-yhteyden tilaa muistissa eivätkä ota siihen kantaa, ne näkevät yhteyksien sijaan vain segmenttejä. (Kurose & Ross, 2013)

TCP segmentti on UDP:hen verrattuna paljon monimutkaisempi johtuen sen tarjoamista takuista. Käymme seuraavaksi läpi niiden tärkeimmät erot.

Järjestysnumero ja kuittausnumero ovat 32-bittisiä kenttiä ja ovat kriittinen osa luotettavan tiedonsiirron toteutuksessa.

Järjestysnumero numeroi jokaisen sovelluskerrokselta tulevan datan tavun. Vastaanottaja käyttää tätä järjestysnumeroa datan järjestämiseen. TCP-protokollan spesifikaatio ei määrittele, mitä kuljetuskerroksen toteutuksen täytyy tehdä, jos se saa viestejä väärässä järjestyksessä. Käytännössä kaikki implementaatiot odottavat puuttuvan datan ja laittavat sen oikeaan paikkaan järjestysnumeroa käyttäen. (Kurose & Ross, 2013)

Vastaanottaja käyttää kuittausnumeroa, kun se lähettää kuittauksia vastaanottamalleen datalle. Yksinkertaistettuna vastaanottaja lähettää takaisin kuittausviestin (ACK), joka on seuraava vastaanottamaton tavu. Esimerkiksi jos se on vastaanottanut 372 tavua niin se lähettää kuittausnumeron 373. Kuittausnumeroa käytetään hyväksi datan uudelleenlähetyksessä. Käytännön toteutuksissa, jos kuittausviestiä ei lähetetä tietyn aikakehyksen sisällä segmentin lähettämisestä, se lähetetään uudestaan. (Kurose & Ross, 2013)

6-bittistä liput-kenttää käytetään ohjaustietoon. URG-, ACK-, PSH-, RST-, SYN- ja FIN-biteillä on omat merkityksensä. Esimerkiksi ACK-bitti on merkitty edellä mainituissa kuittausviesteissä. Loput liput ovat suurimmaksi osaksi yhteyden alustamiseen ja sulkemiseen liittyviä bittejä.

3. HTTP

HTTP/1.1 on kaiken Internet-liikenteen kivijalka sekä kantava voima jokaisessa selainpohjaisessa sovelluksessa. Useiden vuosien ajan on tapahtunut paljon tutkimustyötä isojen yhtiöiden toimesta, joka on ajanut HTTP/2-spesifikaation syntymiseen. Käymme läpi HTTP/1.1- sekä HTTP/2-protokollat ja niiden pääerot. Katsomme myös nopeasti ei standardoituja SPDY- ja QUIC-protokollia joihin HTTP/2-spesifikaatio hyvin vahvasti pohjautuu.

HyperText Transfer Protocol (HTTP) protokolla määritellään Internet Engineering Task Force (IETF) spesifikaatioissa (Internet Engineering Task Force, 1999). HTTP/1.0 määriteltiin vuonna 1996 RFC 1945. HTTP/1.1 versiota aloitettiin määrittelemään samana vuonna ja sen standardointi tapahtui vuonna 1999 RFC 2616 myötä. Seuraava inkarnaatio protokollasta, versio 2 standardoitiin vuonna 2015 (IETF, 2015, *HTTP/2 RFC 7540*). Tämä luku keskittyy HTTP/1.0 ja HTTP/2 vertailuun.

HTTP on yhden palvelimen ja asiakasohjelman, yleisimmin verkkoselaimen välisen kommunikaation protokolla. Spesifikaatiossa määritellään sekä palvelimen että asiakasohjelman odotettu käyttäytyminen kommunikoinnin eri vaiheissa.

3.1. Viestin rakenne

Yksittäinen HTTP-viestitys koostuu asiakasohjelman pyynnöstä (request) ja palvelimen vastauksesta (response). Mekanismi on hyvin yksinkertainen, jolloin sen sovelluskerros on helppo toteuttaa. Monimutkaisuus kasvaa, kun eri otsikkokenttien ja niistä seuraavat operaatiot toteutetaan, kuten asiakasohjelman välimuisti. HTTP-pyynnöstä käytetään yleisesti termiä resurssi tai dokumentti. Tästä syystä HTTP:tä kutsutaan dokumenttipohjaiseksi protokollaksi, koska sen operaatiot kohdistuvat aina yksittäisiin resursseihin. Nykypäivänä Web-sivustot ja palvelinten rajapintapalvelut ovat paljon muutakin kuin pyyntöjä dokumentteihin, mutta yleisesti otettuna dokumenttipohjaisuuden kuvaus on edelleen osuva. (Kurose & Ross, 2013)

Pyyntö koostuu metodista, protokolla versiosta sekä resurssin polusta. Tämän jälkeen tulee erinäisiä pakollisia sekä vapaaehtoisia otsikkokenttiä. Otsikkokentät ohjaavat sekä palvelimen että asiakasohjelman toimintaa. Lopulta tulee viestin data, jonka pituus on määritelty aiemmin otsikkokentällä.

Metodeja on useita. Jokaisella on määritelty viittausta antava käyttötarkoitus: GET, POST, PUT, OPTIONS, HEAD, DELETE, TRACE ja CONNECT. Yleisesti ottaen nykypäivän Web-sivustoilla sekä palvelinrajapinnoissa käytetään resurssien hakuun GET-metodia, resurssin päivittämiseen tai luomiseen PUT- ja POST-metodeja sekä poistamiseen DELETE-metodia. HEAD-metodi on erikoisoperaatio, joka käskee palvelinta lähettämään vain otsikkokentät ilman resurssin dataa. OPTIONS-metodi on myös erikoisoperaatio, jolla pyydetään resurssin asetukset ja säännöt ennen oikeaa pyyntöä. Nykyään se on relevantti osa Cross-origin resource sharing (CORS) mekanismia, jolla voidaan tehdä pyyntöjä eri Internet-domainien välillä. Sen käyttö on yleistynyt erilaisten data-orientoituneiden palvelinrajapintojen räjähdysmäisen kasvun takia. (IETF, 1999, 2015)

3.2. TCP-yhteydet

HTTP/1.1 pyynnöt käsitellään yksi kerrallaan. Jokainen palvelimeen yhdistetty TCP-yhteys voi käsitellä jonossa olevista pyynnöistä yhden kerrallaan. Asiakasohjelma odottaa vastauksen ennen seuraavan pyynnön lähettämistä. (Cohen, Kaplan & Oldham, 1999)

Web-selaimet ja muut asiakasohjelmat ovatkin yleisesti sopineet säännön, että TCP-yhteyksiä otetaan 6-8 verkko-osoitetta kohden, esimerkiksi kun selaat osoitteeseen http://google.com (*Selainten verkkoyhteydet,* 2016*)*. HTTP-spesifikaatio ei ota kantaa siihen, montako TCP-yhteyttä voidaan avata, mutta jos kaikki asiakasohjelmat ottaisivat loputtomasti yhtäaikaisia yhteyksiä, palvelimet eivät kestäisi sen aiheuttamaa kuormaa suosituilla sivustoilla. Huomioitavaa on myös rautatason käytännön rajoitukset. Montako TCP-yhteyttä palvelin ja sen alla oleva käyttöjärjestelmä voi pitää yhtäaikaisesti auki.

Useimmat HTTP/1.0 palvelimet sekä asiakasohjelmat eivät salli TCP-yhteyksien uudelleenkäyttöä. Jokainen yhteys suljettaan yleensä palvelimen toimesta, kun vastaus oli lähetetty (IETF, 1999). Tämä oli yksi isoimpia korjauksia HTTP/1.1 versiossa, missä pyynnön ja vastauksen Connection otsikkokentän oletusarvoksi asetettiin keep-alive, jos palvelin ei erikseen sulje tai pyydä asiakasohjelmaa sulkemaan yhteyttä. Näin ylimääräinen TCP-yhteyden alustaminen saatiin poistettua, kun tiedettiin että resursseja pyydetään paljon lyhyellä aikavälillä. TCP-yhteyden avaaminen ja alustaminen optimaaliseen lähetysnopeuteen on suhteutettuna kallis operaatio erityisesti, jos sitä ei käytetä ison resurssin lähettämiseen. Käytännön toteutukset nykypäivän HTTP-palvelimissa on oletuksena sulkea TCP-yhteys vasta, kun asiakkaalta ei ole tullut uusia pyyntöjä esimerkiksi 15 sekuntiin.

Monien yhteyksien avaaminen ja usein pienten resurssien lähettäminen yksi kerrallaan ei ole optimaalinen mekanismi nykypäivän sovelluksille. HTTP/2-spesifikaatio muuttaa tilannetta merkittävästi. Yksi asiakasohjelma ottaa vain yhden TCP-yhteyden palvelimeen. Tämä muutos on protokollan matalan tason toteutuksessa eikä tule näkymään loppukäyttäjälle, eikä edes Web-sovelluksen kehittäjälle, mutta se vähentää painetta palvelimien TCP-yhteyksien varaamiseen.

HTTP/2 on kanavoitu (multiplex) protokolla, joka poistaa HTTP/1.1 rajoituksen vastauksen odottamisesta ennen uuden pyynnön tekemistä. Asiakasohjelma voi lähettää useamman pyynnön ilman vastausten odottelua. Palvelin voi lähettää vastauksen pyyntöihin haluamassaan järjestyksessä. Yhden resurssin vastaus voidaan myös lähettää useassa osassa sekä sekaisin muiden vastausten kanssa. Multipleksointi sekä demultipleksointi asiakasohjelman päässä takaavat resurssien oikeaoppisen koonnin sekä valmistumisen.

3.3. Otsikkokentät

HTTP/1.1 yksi pääongelmakohta on redundantin datan lähettäminen. HTTP/1.1 on tilaton protokolla, jolloin jokainen vastaus lähettää täydet otsikkokentät. Nykypäivän teknologiakehityksen takia, esimerkiksi keksien (cookie) otsikkokentät vievät paljon tilaa. Yhdistettynä siihen, että pyyntöjen määrä voi olla satoja tai vaativimmissa sovelluksissa tuhansia per sivusto, joko sivun latauksen yhteydessä tai pitkän käyttösession tuloksena, pelkkien otsikkokenttien yhdistetty datamäärä voi kertyä merkittäväksi.

HTTP/1.1 on tekstipohjainen, joka helpottaa ihmisille viestien lukemista ja analysointia. HTTP/2 on binääriprotokolla. Koska viestin data osuus voidaan pakata tehokkaasti jo HTTP/1.1 versiossa, HTTP/2 keskittyi otsikkokenttien koon pienentämiseen. Käyttöön otettiin HPACK otsikkokenttien pakkaus (IETF, 2015, *HPACK RFC 7541*). Pelkistettynä otsikkokenttien teksti on korvattu hyvin pakkautuvilla numeerisilla tyypeillä. Saman HTTP/2-session aikana lähetettyjä otsikkoja sekä niiden arvoja voidaan toistaa vastausten välillä niin, että pelkistetään jo lähetetty arvo tietyllä binääri-sekvenssillä. HPACK on siis suunniteltu vähentämään datan lähetysmäärää pakkaamalla jo nähtyjä otsikkokenttiä hyvin tehokkaasti. Usein session aikana jokainen pyyntö sekä vastaus lähettävät täysin samoja otsikkoja satoja kertoja.

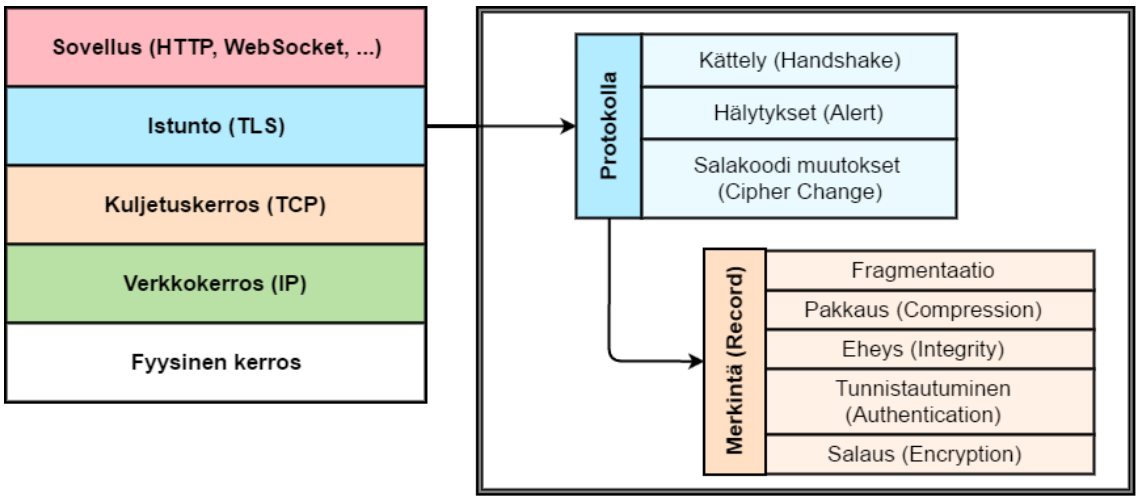
3.4. Salaus

TLS (Transport Layer Security) kehitettiin salaamaan Web-alustan liikenne. Se tarjoaa tunnistautumisen, datan alkuperän validoinnin sekä salakuuntelun suojan. Kun salausta käytetään oikein, välissä olevat toimijat voivat tarkkailla yhteyden toteutumisen, salauksen tyypin, datan lähetyksen frekvenssin sekä approksimaation datan määrästä. Mutta muut kuin alkuperäinen vastaanottaja ja lähettäjä eivät voi lukea tai muokata lähetyksen dataa. Jos lähetystä muokataan, TLS mekanismit havaitsevat sen ja yhteys joko katkaistaan tai muokkauksesta koitetaan toipua. (Grigorik, 2013)

Salaus on tärkeä osa nykypäivän Web-alustaa. Jokaisen sivuston ja palvelun, jossa kulkee henkilökohtaista tietoa, oletetaan käyttävän salausta. Palveluita missä käsitellään osoitetietoja, sähköpostiosoitteita, sosiaaliturvatunnuksia, pankkitietoja tai henkilökohtaisia mediatiedostoja kuten kuvia. Myös viimeaikaiset viranomaisten luvalliset tai luvattomat valtuudet salakuunnella liikennettä lisäävät tuotteiden ja yritysten painetta käyttää nykypäiväistä ja ajantasaista salausta kaikkeen liikenteeseen. Laajamittainen tuotteen adoptio ei tule tapahtumaan, jos tietoturva ei ole ajan tasalla. Nykypäivän teknistä osaamista omaavat kuluttajat ovat tarkkoja valintojensa kanssa.

TLS ja SSL termit tarkoittavat periaatteessa samaa asiaa, mutta käytännössä ne tarkoittavat protokollan eri versioita. SSL 2.0 oli ensimmäinen julkinen versio, mutta se korvattiin nopeasti SSL 3.0 versiolla lukuisten tietoturva-aukkojen löytymisen jälkeen. (Grigorik, 2013)

Kuten näemme kuvasta 1. TLS salauskerros toteutetaan sovelluskerroksessa suoraan TCP-kuljetuskerroksen päällä. Salaus ei muuta HTTP:n perusmekanismeja lainkaan vaan toimii sen alapuolella (Grigorik, 2013). Sen käyttöönotto ei siis vaadi kehittäjältä lähdekoodin muutoksia. Palvelimen ylläpitäjän täytyy asettaa salaukseen käytetyt sertifikaatit käyttöön.



Kuva 1. Transport Layer Security. (Mukautettu lähteestä Grigorik, 2013)

HTTP/2-spesifikaatio ei pakota salatun TLS-yhteyden käyttöä. Mutta mittavat käytännön testit ovat osoittaneet, että HTTP/2-liikenne ei toimi hyvin, jos se ei ole salattua. Tämä johtuu valtavan olemassa olevan Internet-arkkitehtuurin olettamuksesta, että HTTP-liikenne on 1.1 versiota. Esimerkiksi välitys- (proxy) ja välimuistipalvelimet (cache) muokkaavat kyselyjä tai vastauksia niin, että ne voivat rikkoa HTTP/2-liikenteen (Grigorik, 2013). Tämän takia kaikki Internet-selainten kehittäjät, jotka ovat toteuttaneet tai ilmoittanee toteuttavansa HTTP/2-protokollan ovat ilmoittaneet, että HTTP/2 ei ole käytettävissä, jos yhteys palvelimeen ei ole salattu.

Tähän päätökseen on myös toinen syy, yleinen Internet-turvallisuuden parantaminen. HTTP/2 tuo niin suuria parannuksia protokollaan, että se on teknologisesti sekä rahallisen kannattavuuden takia järkevä ratkaisu Web-pohjaisille tuotteille. HTTP/2 adoptiokausi on täydellinen hetki ajaa lisää Internet-liikennettä salatuksi TLS-liikenteeksi.

TLS täysimittainen adoptio ei ole silti onnistunut. Osasyynä on tarvittavien sertifikaattien hinta. Jos haluat palvella itse oman blogisi, et ehkä ole valmis maksamaan salauksesta edes muutamaa kymmentä euroa vuodessa. Vuoden 2015 aikana aloitetun uuden hankkeen Let's Encrypt (*Let's Encrypt*, 2016) tavoitteena on jakaa täysin ilmaisia TLS sertifikaatteja. Mukana on isoja sponsoreita kuten Mozilla, Google, Akamai, Cisco ja EFF. Let's Encrypt jakoi ensimmäiset julkiset sertifikaatit vuoden 2015 lopulla. Marraskuun alussa vuonna 2016 se ilmoitti jakaneensa miljoona sertifikaattia.

Paine Internet-selainten tekijöiltä, kuten Google ja Mozilla, saada kehittäjät salaamaan palvelunsa on myös alkanut ajaa adoptiota. Google Chrome tulee vuosina 2015-2016 sulkemaan pääsyn tiettyihin Web-alustan toiminnallisuuksiin, jos sivusto ei ole salattu, kuten WebRTC ja geopaikannus. WebRTC on videon ja äänen jakamisen mahdollistava teknologia. Sen rajapintojen käytön rajoittaminen ei-salatuissa verkko-osoitteissa laitettiin käytäntöön Google Chrome versiossa 47 joulukuussa 2015. Uskon vahvasti, että Googlen toteutuksen jälkeen tämä trendi tulee siirtymään myös muihin selainten kehittäjiin. Se puolestaan lisää sivustojen kehittäjien yleistä painetta siirtyä salattuun liikenteeseen.

3.5. Protokolla-version neuvottelu

Epäyhteensopivuus HTTP/1.1-protokollan kanssa olisi ollut kuolemanisku HTTP/2 adoptioon asiakasohjelmissa sekä palvelimissa. HTTP/2 ei muuta HTTP-protokollan semantiikkaa millään tavalla. Kaikki ydinkonseptit kuten HTTP-metodit, vastauskoodit, URI:t, otsikkokentät sekä pyyntö-vastaus periaate säilyvät ennallaan (IETF, 2015, *HTTP/2 RFC 7540*). Spesifikaation ei ole järkevää olettaa, että miljoonat olemassa olevat Web-sivustot, sovellukset, palvelimet, palvelut ja kirjastot toteuttaisivat projektinsa uudelleen protokolla päivityksen takia. Miten siis erottaa HTTP/2 kykenevä asiakasohjelma ja palvelin toisistaan tukeakseen läpinäkyvää siirtymistä.

HTTP/1.1-spesifikaatioon kuuluu Upgrade-otsikkokenttä, jota on käytetty pitkään onnistuneesti esimerkiksi WebSocket-yhteyksien päivitykseen. Mekanismi HTTP/2 osalta on saman tapainen. HTTP/2 asiakasohjelma lähettää tämän otsikkokentän pyynnössä. Virallinen HTTP/2 Upgrade-otsikon tunniste on h2. Jos palvelin on samaa mieltä protokollan päivittämisestä se lähettää takaisin 101 Switching Protocols vastauksen. Asiakasohjelman täytyy lähettää myös HTTP/2 asetusten (settings) otsikkokenttä, jossa on lisätietoja yhteisistä pelisäännöistä.

Jotta HTTP/2-yhteys voidaan suojata TLS:n avulla, molempien osapuolien täytyy tukea Application Layer Protocol Negotiation (ALPN) laajennusta (IETF, 2014). Se on TLS:n laajennus, joka suorittaa protokollan sovittelun yhteyden kättelyvaiheessa. ALPN:n säästää yhden kiertoviiveen (round trip) asiakasohjelman ja palvelimen välillä verrattuna normaaliin salauksen kättelyyn. (Grigorik, 2013)

3.6. SPDY ja QUIC

Google julkisti SPDY-verkkoprotokolla vuonna 2009. Toteutus julkaistiin yrityksen Chrome-selaimen versiossa 6 vuonna 2010. Google on iso osa Internet-liikennettä, joten on heidän edun mukaista, että yhteyksiä otetaan mahdollisimman vähän ja lähetetty data on pakattu. SPDY pienensi sivujen latausaikaa sekä paransi tietoturvaa. Se toteutti yhteyden kanavoinnin (multiplexing), viestien priorisoinnin ja datalähetyksen pakkauksen. (Google Research Blog, 2009)

SPDY:tä käytettiin pohjana HTTP/2-spesifikaation tekemisessä ja oli Googlen pitkän kehitystyön tulos. HTTP/2 julkaisun yhteydessä Google ilmoitti poistavansa SPDY-protokollan sen Chrome-selaimesta vuoden 2016 aikana (Chromium Blog, 2015).

Quick UDP Internet Connections (QUIC) oli toinen Googlen merkittävä protokollakokeilu. Sen erottaa SPDY:stä pääasiallisesti se, että se perustuu UDP-kuljetuskerrokseen TCP:n sijasta. Päätavoitteena UDP:n käyttämisessä oli ylimääräisten kiertoviiveiden poistaminen, pakettihävikin minimoinen ja parempi ruuhkautumiskontrolli. (Chromium Blog, 2013)

QUIC on yhä käytössä useissa Googlen palveluissa, eikä sen käytön poistosta ole tehty julkistusta. QUIC:n hyötyjä ei nähtävästi saada paikattua korvaamalla se HTTP/2:lla. Merkittävin QUIC-protokollan käyttäjä on youtube.com, joka on maailman eniten käytetty videosivusto.

3.7. HTTP/2 kriittinen vastaanotto

HTTP/2 julkistaminen on saanut osakseen myös paljon kritiikkiä. Se aloitettiin kopiona SPDY-protokollasta Googlen toimesta vuonna 2012 ja se hyväksyttiin standardiksi alkuvuodesta 2015 (Bakri, Allison, Miller & Oliver, 2015). Tämä on suhteellisen nopea aikajana IETF spesifikaatioksi ja osa kritiikistä on, että se ajettiin poliittisista syistä liian nopeasti läpi. (Kamp, 2015)

HTTP/2 oli hyvä tilaisuus poistaa paljon tietoturvallisuusongelmia aiheuttavat otsikkokeksit (cookie) ja korvata ne asiakasohjelman ja loppukäyttäjän hallitsemalla session tilalla. HTTP/2-binääriprotokolla on saanut myös kritiikkiä tarpeettomasta monimutkaisuudesta sekä epätehokkuudesta. Kamp väittää, että se vaatii enemmän prosessointiresursseja palvelimilta sekä loppukäyttäjien päätelaitteilta. (Kamp, 2015)

SPDY:ltä peritty HTTP-yhteyden tilanpito, vaikkakin sillä saavutetaan monia etuja, taistelee myös HTTP:n perinteistä tilattoman yhteyden mantraa vastaan. Ei salattujen yhteyksien poissulkeminen Web-selaimista nähdään laajalti epäloogisena. Luonnollisesti jokainen yhteys ja datalähetys ei todellisuudessa vaadi kryptografista suojausta, mutta HTTP/2 ja selaintoimittajat pakottavat lähinnä poliittisista syistä TLS-salauksen käytön. (Bakri, Allison, Miller & Oliver, 2015; Kamp, 2015).

4. WebSocket

WebSocket on HTTP-protokollan päälle rakennettu reaaliaikainen ja kaksisuuntainen TCP-yhteys, jolla voidaan välittää binääri- sekä tekstipohjaista dataa. Se määriteltiin parantamaan Web-alustan reaaliaika-kommunikaatiota, jossa HTTP:hen pohjautuvien tekniikoiden latenssit olivat liian isoja. Normaali HTTP-yhteys on pyyntö-vastaus pohjainen, joka ei anna palvelimelle mahdollisuutta lähettää sovellustason viestejä asiakasohjelmaan. WebSocket mahdollista tämän tyyppiset Web-sovellukset. (Grigorik, 2013; Pimentel & Nickerson, 2012; IETF 2011)

Ilman WebSocket rajapintaa kehittäjä voi käyttää kiertomenetelmiä semi-reaaliaikaiseen dataan. Polling- ja long polling -menetelmät ovat hyviä ja yleisesti käytettyjä, mutta ne eivät skaalaudu hyvin, koska ne voivat kuormittaa liikaa resursseja palvelimen puolella. Long polling tekee pyynnön palvelimelle pitkällä tai loputtomalla valmistumisen odotuksella. Palvelin pitää pyyntöä auki niin pitkään, kunnes sillä on uutta dataa. Kun vastaus annetaan, voi se sisältää uuden datan valmiiksi tai asiakas tekee vastauksen palattua uuden kyselyn dataan. Pollaus-menetelmät toimivat kuitenkin optimaalisesti vain, kun palvelimen data päivittyy tasaisin väliajoin. (Wang, Salim & Moskovits, 2013)

WebSocket on yhteensopiva nykyisen HTTP-infrastruktuurin kanssa. Useimmat käänteisvälityspalvelimet (reverse proxy) voivat välittää WebSocket yhteyksiä. Käänteisvälityspalvelin ottaa loppuasiakkaan pyynnöt vastaan ja hakee sen asiakkaan puolesta yhdeltä tai useammalta palvelimelta, välittäen vastauksen asiakkaalle. Näitä palvelimia käytetäänkin usein myös TLS-terminointiin, joka tarkoitta sitä, että salattu WebSocket-yhteys otetaan välityspalvelimeen ja se ottaa salaamattoman yhteyden kohdepalvelimiin. Käytännön säästöt syntyvät siitä, että tämä erikoistunut palvelin voi tehdä TLS-kättelyn nopeammin ja tehokkaammin, sekä sertifikaattien jakaminen palvelininfrastruktuuriin helpottuu. Jos kuljetettava informaatio on arkaluontoista, tätä tekniikka pitäisi käyttää vain, jos kohdepalvelin on samassa sisäverkossa tai samalla fyysisellä koneella. Muuten ensimmäisen yhteyden salaaminen ei luo todellista turvallisuutta. Tämän tason toteutuksen yksityiskohdat ovat tosin kiinni palvelun kehittäjistä eikä loppuasiakkaalla ole paljon mahdollisuuksia varmentaa tilannetta.

4.1. Ohjelmointirajapinta

WebSocket-ohjelmointirajapinta on äärimmäisen yksinkertainen. Avaat yhteyden ja määrität, ovatko viestit binääri- vai tekstipohjaisia. Tämän jälkeen voit kuunnella tulevia viestejä, lähettää viestejä tai sulkea yhteyden. Yksinkertainen alusta ei tarkoita, että sovellukset ovat yksinkertaisia. WebSocket mahdollistaa aivan uudenlaisia elämyksiä Web-alustalle sekä tuo jo olemassa oleviin sovelluksiin reaaliaikaisuutta ja tehokkuutta. (Wang, Salim & Moskovits, 2013)

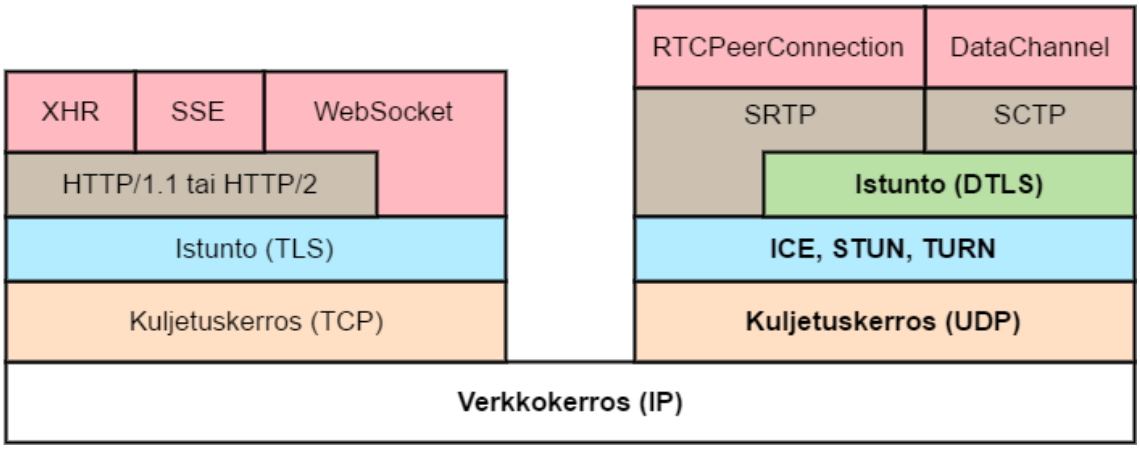
Kun yhteys on muodostettu, sovelluksen kehittäjä saa vapaasti määritellä viestien rakenteen ja niiden käsittelyn sekä asiakkaan että palvelimen päässä. Alustan WebSocket-toteutus vastaa milloin ja minkä kokoisissa osissa viestit lähetetään verkkokerrokselle.

5. WebRTC

Web Real-Time Communication (WebRTC) on yhdistelmä standardeja ja protokollia, jotka mahdollistavat reaaliaikaisen ja keskinäisen (peer-to-peer) äänen, video ja datan välittämisen asiakasohjelmien välillä (Grigorik, 2013). WebRTC assosioidaan pääasiallisesti selainteknologiana, mutta protokolla ei ole rajoittunut vain siihen.

WebRTC:tä kehitetään Web-alustan standardiksi, jonka useat asiakasohjelmat voivat toteuttaa ja kommunikoida keskenään toteutusten välillä (W3C, 2016). Web-alustalla ei ole ollut aikaisemmin mahdollisuutta tähän muuten, kuin erillisten selain-lisäosien kanssa. Jotta tavoite voitiin realisoida, oli eri puolten sovittava ääni- ja videokerroksen käyttämistä koodekeista. Nykyiset toteutukset käyttävät Opus äänikoodekkia ja Google VP8/9 videokoodekkia (Grigorik, 2013). WebRTC tarjoaa myös datan välityksen, jolloin sovelluksen viestien välitykseen ei tarvitse välttämättä aukaista esimerkiksi erillistä WebSocket-yhteyttä.

Reaaliaikainen ääni ja video ei tarvitse TCP:n kuljetuskerroksen takeita, kuten perillemenon varmistamista. Paketteja voi tippua ilman että lopputulos heikentyy huomattavasti. Kuten näemme kuvasta 2, WebRTC valitsi UDP:n kuljetuskerrokseksi. Ero on huomattava muihin mainittuihin teknologioihin verrattuna koska WebRTC ei pohjaudu HTTP-protokollaan.



Kuva 2. WebRTC-protokollan pino (Mukautettu lähteestä Grigorik, 2013)

Kuvan 2. Interactive Connectivity Establishment (ICE) teknologia avustavaa keskinäisen (peer-to-peer) yhteyden muodostamisessa. WebRTC:n alla toimiva RTCPeerConnection on kahden asiakasohjelman välinen yhteys videon ja äänen kommunikointiin. ICE-teknologia avustaa parhaan verkkopolun löytämisessä, jotta yhteys voidaan muodostamaan luotettavasti. Session Traversal Utilities for NAT (STUN) ja Traversal Using Relay NAT (TURN) avustavat verkko-ongelmien kanssa toimimalla välityspalvelimina kun suoraa keskinäistä yhteyttä ei saada muodostettua. Datagram Transport Layer Security (DTLS) salaa liikenteen ja on pakollinen osa WebRTC spesifikaatiota. (Grigorik, 2013)

6. Selainteknologiat

Nykypäivän Web-alusta on hyvin erilainen ja avaa uusia sovellusmahdollisuuksia verrattuna sen alkuperäiseen käyttötarkoitukseen. Dokumenttipohjaiset sivustot ovat edelleen iso osa Internettiä, mutta erikoistuneet sovellukset tuovat paljon enemmän lisäarvoa loppukäyttäjille. Web-alusta takaa yhteensopivuutta ja helppokäyttöisyyttä. Sen kautta sovelluksia on helppo lähestyä, olet vain yhden klikkauksen päässä applikaatiosta. Ei erillisiä latauksia tai asennusta.

6.1. JavaScript

JavaScript on Web-alustan de facto ohjelmointikieli (Mozilla Developer Network, 2016). Se on dynaaminen korkean tason tyypittämätön kieli. Ilman JavaScriptiä mikään nykypäivän sivusto, joka ei ole staattinen dokumentti, ei olisi mahdollinen. Sen vahva assosiaatio Web-alustaan ja selaimiin tekee siitä yhden maailman käytetyimmistä ohjelmointikielistä (RedMonk, 2016). Sen standardi on ECMAScript spesifikaatio, jonka ensimmäinen versio julkaistiin 1997. Viimeisin version 6 ilmestyi vuonna 2015.

JavaScript kielestä puhutaan paljon ja se on hyvin kiistanalainen. Alkuperäinen kehitys tapahtui vain muutamissa viikoissa yhden kehittäjän toimesta. Web alustan räjähtävä suosio veti kielen kiinteäksi osaksi Web-alustaa. Tämä jätti kieleen epäjohdonmukaisuuksia ja paljon paranneltavaa. Standardisoinnin jälkeisiä muutoksia on lähes mahdotonta toteuttaa, koska se rikkoisi taaksepäin yhteensopivuutta (Crockford, 2008).

Yksi kielen pääkehittäjistä, Douglas Crockford, on kirjoittanut kirjan “JavaScript: The Good Parts”. Siinä hän myöntää kielen virheellisyydet ja näyttää sen hyvät osat ja ohjeistaa niiden käyttöön. (Crockford, 2008)

Oikeissa käsissä JavaScript on erinomainen ja joustava kieli. Viime vuosina se on myös siirtynyt selaimista työpöytä- sekä palvelinkäyttöön Node.js ympäristön myötä (*Node.js*, 2016).

ES2015 eli JavaScript versio 6 oli pitkään odotettu päivitys, joka toi paljon uusia primitiivejä, toiminnallisuutta ja tehokkuutta kieleen (Ecma International, 2015). Kielen suorituskyky loppukäyttäjän laitteella paranee lähes jokaisen selainpäivityksen yhteydessä. Selainten kehittäjät panostavat paljon JavaScript-moottorien kehitykseen, koska ne ovat niin osa ihmisten käsitystä siitä mikä selain on paras ja nopein. Isot pelurit moottorikilpailussa ovat Google, Mozilla, Apple ja Microsoft.

6.2. WebGL

WebGL on selainten ohjelmointirajapinta 3D- ja 2D-grafiikalle. Se perustuu OpenGL ES 2.0 (GLES) spesifikaatioon ja tarjoaa samantapaiset renderöintitoiminnallisuudet kuin GLES työpöytäsovelluksissa, mutta HTML ja JavaScript kontekstissa. (Anyuru, 2012)

WebGL olin alun perin Mozilla yhtiön Vladimir Vukićević:n kokeilu vuonna 2006. Useat selaimet toteuttivat tämän jälkeen epästandardeja versioita 3D-grafiikalle. Vuonna 2009 Khronos Group aloitti WebGL standardin kehittämisen (Kaistinen, 2015). Khronos

Group vastaa ja hallitsee myös monista muista laajasti käytetyistä GPU ja 3D-standardeista kuten Vulkan, OpenGL ja OpenCL.

WebGL ohjelmointirajapinta on käytettävissä vain selaimen JavaScript lähdekoodista. Se ei sisällä 3D HTML elementtejä vaan käyttää canvas-elementtiä. Canvas-elementilla voi luoda 3D kontekstin johon tehdään sovelluksen piirtokutsut. Canvas ei ole 3D spesifinen, sillä voi piirtää myös ei WebGL pohjaista 2D-grafiikkaa. WebGL sisällön voi upottaa muun sisällön keskelle tai taustalle vapaasti käyttäen normaaleja Web-tekniikoita kuten CSS-tyylejä. (Kaistinen, 2015)

WebGL on hyvä esimerkki siitä, mihin nykypäivän selainteknologiat pystyvät. 10 vuotta sitten harva olisi voinut kuvitella ajavansa 3D-grafiikkaa ja pelejä selaimessa. Web alustan ylivertainen loppukäyttäjien ulottuvuus tekee siitä kuitenkin houkuttelevan kohdealustan kehittäjille.

Applikaation jakelu on helppoa ja halpaa. Päivitysten tekeminen on helppoa. Web-selaimet toimivat kaikilla merkittävillä alustoilla (Windows, Mac OS, Linux, Android, iOS). Web-alusta on myös ilmainen, yleisimmät ammattimaiset 3D- ja pelimoottorit ovat maksullisia. (Anyuru, 2012)

WebGL 2.0 version on parhaillaan kehityksen alla. Se tulee pohjautumaan työpöytäympäristön Open GL ES 3.0 versioon. WebGL 2.0 ei tule olemaan täysin yhteensopiva WebGL kanssa. Sen tulee parantamaan suorituskykyä sekä tuomaan uusia ominaisuuksia. Osa Web-selaimista tukee jo osittain 2.0 versiota. (Khronos Group, 2016)

6.3. WebComponents

WebComponents termin alle lukeutuu useita teknologioita. Korkean tason käsitteenä WebComponents tarkoittaa uudelleenkäytettäviä HTML-elementtejä. Web-selain toteuttaa nämä standardit jolloin kehittäjä ei tarvitse erillisiä kirjastoja niiden käyttämiseen.

Web-alustalla on ollut historiallisesti vaikeuksia yhdistää HTML-, JavaScript- ja CSS-teknologia konkreettiseksi paketiksi. Sovellukseen rakennettujen HTML-elementtien uudelleenkäyttö on ollut monimutkaista. Jos haluat saman elementin eri puolille sivuasia, on sinun täytynyt joko kopioida koko HTML-koodi tai käyttää kirjastoja jotka osaavat monistaa rakenteen JavaScript-koodista.

Kun kehität sovelluksen WebComponents periaatteella, luot elementin kerran. Tämä WebComponent-elementti sisältää HTML-, JavaScript- ja CSS-koodin pakettina yhdessä tai useammassa tiedostossa. Vois monistaa elementtiä eri parametreilla sovelluksessasi vaivattomasti.

Viralliseksi standardiksi asti on ehtinyt vasta HTML Templates. Se määrittelee, miten uudelleenkäytettävät elementit asetetaan selaimen DOM-puuhun. (W3C, 2014). Muut spesifikaatiot ovat vielä työn alla, mutta ovat edistyneet hyvin vuosien 2015-2016 aikana. Useat selaimet ovat toteuttaneet jo tärkeimmät WebComponent-teknologiat. Sovelluskehittäjät voivat käyttää niitä ehdollisten apukirjastojen (polyfill) avulla. (*Can I use WebComponents*, 2016)

7. Yhteenveto

Verkkoprotokollat ovat kehittyneet muutaman viime vuoden aikana merkittävästi. Eritoten Web-alustan vahvasti hyödyntämät HTTP/2, WebSocket ja QUIC ovat tuoneet uutta virtaa sovelluskehittäjille. Muutos aiempaan on merkittävä, koska Web-alustan protokollat polkivat paikallaan vuosikymmenen. WebSocket saapui ensin tarjoamaan reaaliaikaiset ja kaksisuuntaiset yhteydet, joka mahdollisti täysin uudenlaisia ja nopeavasteisia sovelluksia. Rinnalla Web-alustan jätit kehittivät ilman standardisointia SPDY- ja QUIC-protokollat, jotka toimivat vahvana pohjana HTTP/2-protokollan kehityksessä.

HTTP/2 on mielestäni selvästi merkittävin muutos Web-alustan nopeuden, tehokkuuden ja turvallisuuden kehityksessä. Ennen sitä sovelluskehittäjät joutuivat kehittelemään kiertototeutuksia HTTP/1.1 rajoitteiden takia. Tämä johti koko Web-ekosysteemin monimutkaisuuden hienovaraiseen kasvuun sovelluskehittäjän näkökulmasta. HTTP/2 tuo alla toimivan verkkoprotokollan tälle vuosisadalle, jolloin Web-sivustot eivät ole enää vain staattisia dokumentteja.

Selainteknologioiden puolella merkittävimmät harppaukset ovat tehneet selaintoimittajat. Niin sanottujen aina tuoreiden Web-selainten suosio on kasvanut räjähdysmäisesti. Microsoft Internet Explorer ei ole enää maailman eniten käytetty selain. Tämä parantaa sovelluskehittäjien lähtötilannetta, kun he tekevät teknologiavalintoja uusiin projekteihin. Nykypäivänä kehittäjät voivat luottaa paljon paremmin, että tuoreetkin selainteknologiat ja standardit ovat toteutettu käyttäjien Web-selaimissa. JavaScript versio 6 mahdollistaa paremmin isojen sovellusten jäsentelyä sekä poistaa käsin kirjoitetun avustuslähdekoodin tarvetta. Kaikki tämä mahdollistaa nopeamman ja tehokkaamman sovelluskehityksen.

Selainten välinen kilpailu verkkoprotokollien, selainteknologioiden sekä JavaScript moottoreiden toteutuksessa on kovaa. Tämä kilpailu tuo nopeutta ja luotettavuutta Web-selainten loppukäyttäjille ja yhtenäisemmän Web-alustan sovelluskehittäjille.

Tutkimukseni pohjalta voimme esittää, että elämme Web-alustan kulta-aikaa. Kokonaiskuvassa merkittävimpien HTTP- ja JavaScript-teknologioiden seuraavat versiot ovat jo työn alla. Molemmat työryhmät ovat antaneet ymmärtää, että seuraavia uudistuksia ei tarvitse odottaa vuosikymmentä. Web-alustan tulevaisuus näyttää valoisalta.

Lähteet

Anttila, A., (2000). *TCP/IP-tekniikka* (1. painos). Helsinki Media. WSOY.

Anyuru, A. (2012). *Professional WebGL Programming: Developing 3D Graphics for the Web* (2. painos). Wrox.

Bakri, H., Allison, C., Miller, A., & Oliver I. (2015) HTTP/2 and QUIC for Virtual Worlds and the 3D Web. *Procedia Computer Science 56*, 242-251.

Can I use WebComponents. (2016). Haettu 12.5.2016, saatavilla http://caniuse.com/#search=components

Chromium Blog. (2013). *Experimenting with QUIC*. Haettu 26.4.2016, saatavilla: http://blog.chromium.org/2013/06/experimenting-with-quic.html

Chromium Blog. (2015). *Hello HTTP/2, Goodbye SPDY*. Haettu 26.4.2016, saatavilla: http://blog.chromium.org/2015/02/hello-http2-goodbye-spdy.html

Cohen, K., Kaplan, H., & Oldham J. D. (1999). *Managing TCP Connections under Persistent HTTP*. Elsevier B.V.

Crockford, D. (2008). *JavaScript: The Good Parts*. O'Reilly Media.

Ecma International. (2015). *ECMAScript® 2015 Language Specification.* Haettu 26.4.2016, saatavilla: http://www.ecma-international.org/ecma-262/6.0/ECMA-262.pdf

Google Research Blog. (2009). *A 2x Faster Web*. Haettu 26.4.2016, saatavilla: http://googleresearch.blogspot.fi/2009/11/2x-faster-web.html

Grigorik, I. (2013). *High Performance Browser Networking*. O'Reilly Media.

IPv6. (2016). Haettu 25.4.2016, saatavilla: https://www.nro.net/ipv6

IETF. (2015). *HPACK RFC 7541*.

IETF. (1999). *HTTP/1.1 RFC 2616*.

IETF. (2015). *HTTP/2 RFC 7540*.

IETF. (2011). *The WebSocket Protocol RFC 6455*.

IETF. (2014). *Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension RFC 7301*.

Kaistinen, M. (2015). *Open source solutions for multiplatform graphics programming*. Lappeenrannan University of Technology.

Kamp, P-H. (2015). HTTP/2.0: the IETF is phoning it in. *ACM Queue Volume 13 Issue 2, February 2015*.

Khronos Group. (2016). *WebGL 2 Specification Editors Draft*. Haettu 24.4.2016, saatavilla: https://www.khronos.org/registry/webgl/specs/latest/2.0/

Kurose, J. F., & Ross, K. W. (2013). *Computer Networking: a top-down approach* (4. ja 6. painos). Pearson Education, Inc.

Let's Encrypt. (2016). https://letsencrypt.org/

Mozilla Developer Network. (2016). *JavaScript*. Haettu 26.4.2016, saatavilla: https://developer.mozilla.org/en-US/docs/Web/JavaScript

Node.js. (2016). https://nodejs.org

Pack, S., Hong, E., Park, I., Kim, J. J., Ko, D. (2002). Game Transport Protocol: A Reliable Lightweight Transport Protocol for Massively Multiplayer On-line Games (MMPOGs). *Proc. SPIE 4861, Multimedia Systems and Applications V, 83.*

Pimentel, V., & Nickerson, B. G. (2012). Communicating and Displaying Real-Time Data with WebSocket. *Internet Computing, IEEE, July/August*, 45-53.

RedMonk. (2016). *The RedMonk Programming Language Rankings: January 2016*. Haettu 23.4.2016, saatavilla: http://redmonk.com/sogrady/2016/02/19/language-rankings-1-16/

Selainten verkkoyhteydet. (2016). http://www.browserscope.org/?category=network

Wang, V., W., Salim, F., & Moskovits, P. (2013). *The Definitive Guide to HTML5 WebSocket*. Apress Media LLC.

W3C. (2014). *HTML Templates.*

W3C. (2016). *WebRTC 1.0 Working Draft.*