

Selainsovelman ja palvelimen välinen viestintä

Bemmu Sepponen

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Ohjaaja: Erkki Mäkinen
18.12.2013

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
Bemmu Sepponen: Selainsovelman ja palvelimen välinen viestintä
Pro gradu -tutkielma, 65 sivua
Joulukuu 2013

Tiivistelmä

Tutkimuksella pyritään tarjoamaan WWW-ohjelmoinnista kiinnostuneelle lukijalle yleiskuva viestien välittymisestä selaimesta WWW-palvelinohjelman kautta palvelinsovelmalle. Esitellään useita saatavilla olevia kommunikointitekniikoita, tarjoten yleiskatsaus saatavilla olevista rajapinnoista ja protokollista ohjelmakoodiesimerkein. Selainten sisäänrakennettujen viestintäkeinojen lisäksi tarkastellaan suosittujen lisäosien Javan ja Flashin tarjoamia mahdollisuuksia. Esitellään myös jatkuvien yhteyksien käyttöä, jotka tarjoavat yksittäisiin HTTP-kutsuihin verrattuna resursseja säästäviä ja vähälatenssisempia viestintäkeinoja.

SISÄLLYS

1	Johdanto	1
2	Taustaa	3
2.1	TCP/IP	3
2.2	DNS-nimipalvelujärjestelmä	3
2.3	URL	4
2.4	HTML-lomakkeet	6
3	HTTP	9
3.1	GET	9
3.2	POST	10
3.3	Muut pyyntötyypit	13
4	HTTP-kutsujen välittyminen WWW-palvelimelta palvelinsovelmalle	15
4.1	Palvelinsovelman käynnistäminen yksittäisen kutsun palvelimiseksi CGI:llä	16
4.2	Kutsun välittäminen jo käynnissä olevalle palvelinsovelmalle FastCGI:llä	16
4.3	SCGI	17
4.4	Kutsun välittäminen Python-palvelinsovelmalle WSGI:llä	18
4.5	Sovelman sisällyttäminen WWW-palvelinohjelmaan	19
4.6	Kutsun tulkinta palvelinsovelmassa	20
5	Tiedonsiirtomuotoja	22
5.1	JSON-tiedonsiirtomuoto	22
5.2	JSONin jäsentäminen valmiilla kirjastoilla	24
5.3	Protokollabufferit	24
5.4	Tiedostojen lähettäminen lomakkeella	26
5.5	Muita tiedonsiirtomuotoja	27
6	Comet-yhteydet JavaScript-sovelmista	28
6.1	Kiertokysely XMLHttpRequest-oliolla	28
6.2	Pitkäkiertokysely	29
6.3	Ikuinen iframe	30
6.4	XMLHttpRequest multipart/x-mixed-replace	32
6.5	XMLHttpRequestin responseType-attribuutin valvonta	33
6.6	EventSource API	35
6.7	WebSocket	37
6.8	WebRTC	42
7	Yhteydet Java-sovelmista	46

7.1	Javan vastakkeet (sockets)	47
7.2	JDBC	47
7.3	RMI	48
8	Yhteydet Adobe Flash -sovelmista	51
8.1	Flashin loadVariables-funktio ja URLLoader-luokka	51
8.2	Vastakkeet ActionScriptissä	52
8.3	Adobe Cirrus	54
9	Yhteenveto	56
	Viiteluettelo	57
10	Lyhenteet	65

1 JOHDANTO

WWW-kehityksen alkuaikoina sivut olivat staattisia kokonaisuuksia, joiden kanssa ainut interaktio oli sivujen välillä navigointi linkkien kautta. Seuraavaksi tulivat mukaan lomakkeet ja evästeet, joiden avulla voitiin luoda dynaamisia WWW-palveluita. Käyttäjät pystyivät vaikuttamaan sisältöön muun muassa kirjoittamalla viestejä vieraskirjoihin tai lähettämällä lomakkeen avulla palautetta sivuston ylläpitäjille. Sessioavainten ansiosta käyttäjän identiteetti pysyi tiedossa siirtyessä sivulta toiselle, joten sisältö voitiin räätälöidä käyttäjäkohtaisiksi.

Luvussa 2 kerrotaan hieman tarvittua taustatietoa. Luku alkaa TCP/IP-protokollaperheen kuvauksella, joka on kaikkien tutkielmassa esiteltyjen kommunikointitapojen taustalla. Seuraavaksi tutustutaan lyhesti DNS-nimipalvelujärjestelmään, URLeihin ja HTML-lomakkeisiin, jotka ovat myös tällaisten palveluiden tärkeitä rakennuspalikoita. Luvussa 3 kuvaillaan mitä palveluiden toteuttamiseen käytetyt GET-, POST- ja uudemmat muuntotyypiset kutsut tarkoittavat.

WWW-palveluiden toteuttamiseksi on selaimen näkökulman lisäksi ajateltava myös palvelimen toimintaa. Palvelimella on ajettava palvelun toteuttavaa palvelinsovelmaa, jonka on jollakin tavalla saatava kutsut ja vastattava niihin. Välissä on lähes poikkeuksetta myös WWW-palvelinohjelma, joten tietojen on jollakin tavalla välityttävä WWW-palvelinohjelmalta palvelinsovelmalle. Tietojen välitykseen on kehitetty useita tapoja, jotka poikkeavat toisistaan erityisesti niiden tehokkuuden näkökulmasta. WWW-palvelimen ja palvelinsovelman välisiin kommunikointitapoihin perehdytään luvussa 4.

Lukujen 3 ja 4 tietojen perusteella voidaan ymmärtää, miten viestejä välitetään perinteisissä dynaamisissa WWW-sovelluksissa selaimen ja palvelinsovelman välillä. Tiedoilla täytyy kuitenkin olla jokin sarjoitusmuoto, jotta rakenteinen tieto voidaan sarjoittaa (englanniksi ”serialize”) selainsovelmassa tavujonoksi ja edelleen palauttaa rakenteiseksi tiedoksi palvelinsovelmassa. Nykyisin suosituin vaihtoehto tähän lienee JSON-esitysmuoto, joka esitellään luvussa 5 muiden tiedonsiirtomuotojen ohella.

JavaScript-komentosarjakielen uuden HTTP-kutsut mahdollistavan XMLHttpRequestn avulla tuli mahdolliseksi lähettää selaimesta tietoa palvelimelle myös sivulatausten välillä, jolloin voidaan luoda käyttäjätunnumaltaan työpöytäsovelluksia muistuttavia ratkaisuja ilman selainlisäosia. Olion ansiosta esimerkiksi pikaviestintäsovelluksesta viestien lähettäminen on mahdollista ilman, että selaimen tarvitsi ladata koko sivua uudelleen. Muilta käyttäjiltä tai palvelimelta saapuvien viestien vähäviiveinen vastaanottaminen oli kuitenkin edelleen heikkoutena.

XMLHttpRequest-oliolla voidaan toki kiertokysellä palvelinta uudelleen ja uudelleen muuttuneiden tietojen vastaanottamiseksi, mutta tehokkaamman viestinnän kannalta olisi parempi, jos jollakin tavalla uusista viesteistä saataisiin heti tieto ilman turhia kutsuja. Ohjelmistokehittäjät ovat keksineet useita tapoja resursseja säästävään ja vähälatenssisempaan viestintään käyttäen hyväksi selainten sellaisiakin ominaisuuksia, jotka eivät olleet alunperin tarkoitukseen suunniteltuja. Toisaalta myös selainkehittäjät ovat vähitellen lisänneet selaimiin toinen toistaan parempia tapoja toteuttaa tällaisia yhteyksiä. Näihin comet-tekniikoiksi kutsuttuihin viestintäkeinoihin perehdytään luvussa 6. Luvun lopussa esitellään myös uusi WebRTC-tekniikka, jonka avulla videokonferenssisovellusten tai UDP-yhteyksiä vaativien pelien toteuttaminen JavaScriptistä tuli mahdolliseksi.

Monet JavaScriptistä käytettävissä olevat nykYTEKNIIKAT olivat jo aikaisemmin mahdollisia selainlisäosia hyödyntämällä. Vaikkakin nykyään on entistä suosittumpaa toteuttaa selainsovelma mahdollisuuksien mukaan vaatimatta lisäosien käyttöä, voi niiden käyttö edelleen joissakin tilanteissa olla tarpeen. Erityisesti kun halutaan avata vapaita sovikeyhteyksiä tai toteuttaa videokonferenssiratkaisuja mahdollisimman laajalle käyttäjäkunnalle tulee myös selain lisäosien huomiointi aiheelliseksi. Selainlisäosien tarjoamiin mahdollisuuksiin tutustutaan Javan osalta luvussa 7 ja Flashin osalta luvussa 8. Lopulta tarjotaan yhteenve-to ja annetaan suosituksia eri tarkoituksiin laadittujen sovelmien toteutuksesta luvussa 9.

2 TAUSTAA

2.1 TCP/IP

TCP/IP on kokoelma protokollia, jotka mahdollistavat luotettavan kommunikoinnin laitteiden välillä. Siirrettävät tiedot pilkotaan paketeiksi, jotka reititetään IP-osoitteiden perusteella. Yksinkertaistaen jokaisella verkon laitteella on oma IP-osoitteensa, jonka perusteella paketit toimitetaan oikealle vastaanottajalle. Internet on vikasietoinen, koska tieto hajotetaan pienempiin paketteihin, jotka voivat päätyä vastaanottajalle useita mahdollisia reittejä pitkin [Lintula 2004, 3].

Jokainen laite tietää oman IP-osoitteensa lisäksi käyttämänsä nimipalvelimen IP-osoitteen, aliverkonpeitteen ja oletusyhdyskäytävän IP-osoitteen. Aliverkonpeite kertoo lähettävälle laitteelle, mitkä IP-osoitteet ovat omassa aliverkossa. Jos aliverkonpeitteen perusteella pakettia ei voida lähettää suoraan, se välitetään oletusyhdyskäytävälle. Näin Internet koostuu pienemmistä verkoista, jotka ovat yhteydessä toisiinsa [Lintula 2004, 3].

Nimipalvelimen IP-osoite on tarpeen, kun halutaan muuntaa käyttäjäystävällisemmät verkkonimet IP-osoitteiksi. Esimerkiksi ”example.com” voi muuttua IP-osoitteeksi ”93.184.216.119”.

Sovikkeet (englanniksi ”socket”) tarjoavat ohjelmille hyödyllisen abstraktion, jossa ne voivat avata yhteyden toiseen sovikkeeseen. Tietoa voidaan sitten kirjoittaa sovikkeeseen ja lukea sieltä, ilman että jokaisen sovikkeita käyttävän ohjelman täytyisi toteuttaa tiedon jakoa paketteihin ja pakettien koostamista jälleen tietovirraksi vastaanottajan puolella [RFC793, 7].

TCP rakentaa IP-protokollan päälle luotettavamman yhteystavan takaamalla, että paketit tulevat vastaanottajalle pyytämällä vastaanottajalta kuittauksen (ACK) niiden saapumisesta [RFC793, 4]. Jokaisen paketin jälkeen ei kuitenkaan ole tarpeen odottaa kuittausta, vaan TCP:ssä on kasvava lähetysikkuna. Lähetysikkunan koon mukaan voidaan lähettää useampia paketteja kerralla ennen kuittauksen odottamista [Lintula 2004, 4]. Sen avulla vastaanottaja voi myös säädellä saapuvan datan määrää [RFC793, 4].

2.2 DNS-nimipalvelujärjestelmä

Yleensä käyttäjä viittaa verkkosivuihin verkkonimen perusteella. Yhteyden avaamista varten on kuitenkin tiedettävä numeerinen IP-osoite. Täytyy siis olla tapa, jolla verkkonimi muunnetaan IP-osoitteeksi. Tätä tapaa kutsutaan nimipalvelujärjestelmäksi (englanniksi ”Domain Name System”).

Internetin alkuaikoina kaikkien nimien numeeriset osoitteet olivat saatavilla yksittäisessä tiedostossa. Ratkaisuna se oli helppo ymmärtää. Haetaan vain kyseinen tiedosto, josta käy heti ilmi haetun verkkonimen IP-osoite. Internetin kasvaessa ratkaisu ei kuitenkaan enää ollut riittävän tehokas, joten vuonna 1983 esiteltiin uusi hajautettu nimipalvelinjärjestelmä [RFC882]. Hajautetussa järjestelmässä yksittäisen listan sijaan on useita nimipalvelimia, joilla on vastuu tietyistä nimialueista. Nimipalvelimet voivat etsiä vastauksia kyselyihin muilta nimipalvelimilta, joten listan sijaan nykyinen järjestelmä on puurakenne.

Vaikkakin nimien selvittäminen on käyttöjärjestelmän tarjoama palvelu, nykyaikaiset selaimet joutuvat jossain määrin ottamaan kantaa DNS-selvityksiin, koska niiden ennakoiminen nopeuttaa selaimen toimintaa. Esimerkiksi Chrome-selaimessa on esiselvitysominaisuus (”DNS Prefetch”), joka ennakoiden hakee nimien IP-osoitteet.

2.3 URL

URL-merkkijonoja (”Uniform Resource Locator”) käytetään osoittamaan WWW-sivuja [RFC3986]. Esimerkki URL:sta on edellä mainittu ”http://example.com”. URL koostuu skeemasta ja siitä riippuvasta sisällöstä kaksoispisteellä eroteltuna. Esimerkissä skeema on ”http” ja sisältö on ”//example.com”. Selainohjelmoinnin tapauksessa ollaan lähes aina tekemisissä skeemojen ”http” tai ”https” kanssa, joten perehdytään tähän tapaukseen. Selainohjelmoinnissa välitetään myös usein tietoa URLien kautta selaimen ja palvelimen välillä sivun lataamisen jälkeenkin.

HTTP- ja HTTPS-skeemaisen URLin osat ovat omistaja, polku ja toiminto: ”http://example.com/polku?toiminto”. Polku kertoo haetun resurssin, kun taas toiminto voi sisältää tietoa, jota mahdollisesti palvelimella ajettava sovelma voi käyttää. Toiminto voi vaikuttaa palvelimen palauttamaan tietoon. Polku voi esimerkiksi sisältää hakukoneen hakukyselyn. Koska kysymysmerkki erottaa polun ja toiminnon toisistaan, ei itse polussa voi sekaannuksen välttämiseksi käyttää kysymysmerkkiä. Tällaisia erotinmerkkejä on kysymysmerkin lisäksi muitakin. Erotinmerkkejä kutsutaan varatuiksi merkeiksi.

2.3.1 HTTP-URLien koodaus

Helposti ymmärrettävä esimerkki kontekstin vaikutuksesta koodattaviin merkkeihin on välilyönnin käsittely [Épardaud]. Jos ilmaistavassa polku-osassa esiintyy plus-merkki, sitä ei ole välttämätöntä koodata. Listauksessa 2.1 esitetään miten plus-merkki koodaantuu sen esiintyessä polkuosassa.

Listaus 2.1: Plus-merkki polussa

```
Skeema: http
Omistaja: example.com
Polku: C++
Toiminto: toiminto

http://example.com/C++?toiminto
```

Toisaalta taas jos ilmaistavassa toiminto-osassa esiintyisi sama plus-merkki, tulee se prosenttikoodata muotoon %2B. Sama merkki voi siis tilanteesta riippuen vaatia erilaista esitystapaa. Listauksessa 2.2 esitetään miten plus-merkki koodaantuu sen ollessa toiminto-osassa.

Listaus 2.2: Plus-merkki toiminnossa

```
Skeema: http
Omistaja: example.com
Polku: polku
Toiminto: C++

http://example.com/polku?C%2B%2B
```

Ei ole mahdollista koodata kokonaista URL-merkkijonoa oikein tietämättä sen erottelua osiin. Esimerkiksi omistajan ollessa ”example.com” ja polun ollessa ”a?b=c” olisi väärin yhdistää ne ensin merkkijonoksi ja koodata vasta sitten.

”http://example.com/a?b=c” ei enää kertoisi, halutaanko nyt ilmaista polku ”a” ja toiminto ”b=c”, vaiko kenties polku ”a?b=c”. Oikea koodaustapa riippuu kontekstista ja merkin luokasta. Merkit voidaan jakaa kolmeen luokkaan:

- Varatut merkit
- Varaamattomat merkit
- Muut merkit

Varatut merkit voi URLin osasta riippuen näyttää sinällään, tai ne on prosenttikoodattava (englanniksi ”percent-encoding”). Varattuja merkkejä kutsutaan ”varatuiksi”, koska niitä saatetaan skeemasta riippuen käyttää erotinmerkkeinä. *Varaamattomat merkit* koostuvat numeroista 0-9, kirjaimista a-zA-Z ja merkeistä viiva, alaviiva, piste ja tilde (- _ . ~). [RFC3986, 2.3.]. Varaamattomat merkit voidaan aina jättää koodaamatta. *Muut merkit* on aina prosenttikoodattava. Prosenttikoodauksessa tavu esitetään muodossa %XX, jossa XX on tavun heksadesimaalinen esitys [RFC3986, 2.1.].

Itse skeeman nimeä (esim. ”http”) ei ole tarpeen koodata, sillä skeema valitsee käytettävän protokollan, eikä se siten sinällään merkkijonona siirry palvelimelle. Viestinnän kannalta on hyödyllistä tietää, että URLeja ei useinkaan voi muodostaa oikein vain yhdistämällä merkkijonoja, sillä niiden osat on tilanteesta riippuen koodattava tietyllä tavalla. On varsin hankalaa selvittää, mitkä merkit missäkin yhteydessä on koodattava, joten on parempi käyttää valmiita kirjastoja URLien muodostamiseen.

Mahdollisesti yleisin URLeihin liittyvä tehtävä WWW-ohjelmoinnissa on koostaa URL sellaisessa tilanteessa, kun sen muut osat toimintoa lukuunottamatta ovat tiedossa. Seuraavassa pari esimerkkiä tästä tapauksesta WWW-ohjelmoinnissa suosituilla Python- ja PHP-ohjelmointikielillä. Esimerkkinä listauksessa 2.3 esitetään miten toiminto-osio muodostetaan urllib-pakettia käyttäen Python-ohjelmointikielessä.

Listaus 2.3: Toiminto-osion muodostaminen Python-ohjelmointikielellä

```
import urllib
parametrit = {'b' : 'c'}
toiminto = urllib.urlencode(parametrit)
url = 'http://example.com/foo?' + toiminto
```

Listauksessa 2.4 esitetään miten PHP-kielisessä ohjelmassa on parametrin avain- ja arvo-osuudet koodattava erikseen, mutta toisaalta tarvittava koodausfunktio löytyy valmiina.

Listaus 2.4: Toiminto-osion muodostaminen PHP-ohjelmointikielellä

```
<?php
$parametrit = array("b" => "c");
$url = 'http://example.com/foo?';
foreach ($parametrit as $key => $value) {
    $url .= rawurlencode($key) . '=' . rawurlencode($value);
}
print $url;
?>
```

2.4 HTML-lomakkeet

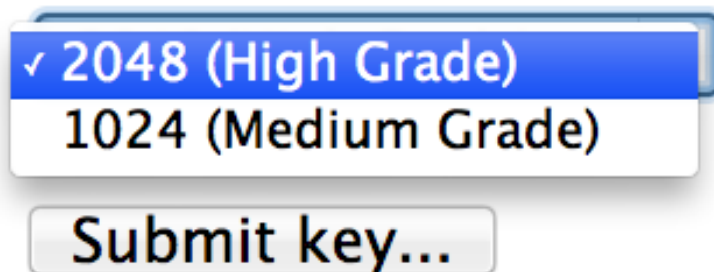
Lomake on WWW-sivun osa, joka voi sisältää useita lomakekenttiä [HTML5Nightly 4.10.1]. Lomakekenttiä ovat esimerkiksi tekstikentät, painikkeet ja valintaruudut (englanniksi ”checkbox”). Lomakkeella kerätään palvelimelle lähetettäväksi tarkoitettuja tietoja.

Lomakkeita laaditaan HTML-koodissa sisällyttämällä lomakekenttiä edusta-

via tageja form-tagien sisään. Tällöin voidaan myös sanoa, että kyseinen lomake omistaa listatut lomakekentät. Lähetettävän lomakkeen sisältöön vaikuttavat tagit ovat `<button>`, `<input>`, `<keygen>`, `<object>`, `<output>`, `<select>` ja `<textarea>` [HTML5Nightly 4.10.2]. Näistä yleisimpiä ovat painiketta edustava `<button>`, type-attribuutista riippuen useita erilaisia tietotyyppettä edustava `<input>`, annetuista vaihtoehdoista valitsemisen mahdollistava `<select>` ja vapaatekstilaatikko `<textarea>`. Tutustutaan tarkemmin vähemmän tunnettuihin tageihin `<keygen>`, `<object>` ja `<output>`.

2.4.1 Keygen

Epäsymmetrisessä julkisen avaimen salauksessa tarvitaan yksityinen ja julkinen avain. Lomakekenttää `<keygen>` hyödyntäen voi selain luoda tällaisia avainpareja. Kuvassa 2.1 `<keygen>`-tagin ulkoasu selaimessa. Yksityinen avain tallentuu käyttäjälle, kun taas julkinen avain lähetetään osana lomaketta. Jos keytype-attribuutilla ei muuta pyydetä, on avainten oletustyyppi RSA.



Kuva 2.1 `<keygen>` selaimessa

Käytännössä keygen-lomakekenttää ei juuri voi hyödyntää, sillä se ei lainkaan toimi kirjoitushetkellä saatavilla olevissa Internet Explorer -selaimissa (uusin 11.0.9), joilla on kuitenkin suuri markkinaosuus. On epätodennäköistä, että tukea tulevaisuudessakaan selaimen tulee, sillä Microsoftin Internet Explorer -selainta kehittävä ryhmä ei ole sen kannalla. ”On erittäin epätodennäköistä, että Microsoft koskaan toteuttaa `<keygen>`-tukea. Mielestämme se ei ole tärkeä asiakkaillemme.”, kirjoitti IE-ryhmän projektivastaava W3C:n sähköpostilistalle syyskuussa 2009 [W3C Mailing List].

2.4.2 Object

Object-tagia käytetään kun WWW-sivulle halutaan sisällyttää liitännäisiä. Yleisin syy <object>-tagin käyttöön on Flash-lisäosan sisällyttäminen sivulle. On vähemmän tunnettua, että <object>-tagia voi käyttää myös lomakkeissa. Kun lomakkeessa on liitännäiseen viittaava <object>-tagi, kysyy selain lomakkeen lähettämisen yhteydessä myös liitännäiseltä, onko sillä tietoja, joita se haluaa tarjota lähtevän HTTP-pyyntön mukaan [HTML5Nightly 4.10.22.4].

2.4.3 Output

Kun lomakkeella on kenttiä, joiden perusteella johdetaan muu kenttä, voidaan lopputulosta edustavaa kenttää ilmaista <output>-tagilla. Output-tagin eroaa input-tagista siinä, että vaikka sen arvoon voi viitata kuten input-taginkin ja muutunut arvo näkyy käyttäjälle, ei käyttäjä kuitenkaan pääse muokkaamaan sen sisältöä.

Jos esimerkiksi verkkokaupan sivuilla olisi lomake, joka sisältää ostettavat tuotteet sekä painikkeen jolla niiden yhteishinta voidaan laskea, voisi yhteishinta olla <output>-tagissa, kuten listauksen 2.5 esimerkissä on esitetty. Loppuhinta päivittyy automaattisesti, kun käyttäjä muuttaa input-kenttien sisältöä.

Listaus 2.5: Output-tagin käyttö

```
<form onsubmit="return false"
  oninput="o.value = puvut.valueAsNumber*10000 + polttoaine.
    valueAsNumber*10">
Avaruuspukuja <input value=1 name=puvut type=number> kappaletta<
  br>
Polttoainetta <input value=100 name=polttoaine type=number>
  litraa<br>
Loppuhinta: <output name=o></output> euroa.
</form>
```

3 HTTP

3.1 GET

”Hypertext Transfer Protocol” eli hypertekstin siirtoprotokolla [Wikipedia:HTTP] on käytäntö, jolla asiakasohjelma saa haluamansa dokumentin tai voi lähettää haluamansa tiedon WWW-palvelimelle.

Ensimmäinen HTTP-versio oli HTTP/0.9. Internetin protokollat määritellään yleensä RFC-dokumenteissa (”Request for Comments”). RFC:t ovat joukko asiakirjoja, jotka kuvailevat Internetin käytäntöjä [Korpela]. HTTP/0.9:lle ei ole RFC-dokumenttia, mutta siitä on saatavilla vuonna 1991 julkaistu lyhyt yksinkertainen kuvaus verkossa [HTTP/0.9]. Vuonna 1992 tuli saataville myös tarkempi kuvaus [HTTP1992]. Alussa HTTP:llä ei ollut lainkaan versionumeroa, vaan sitä alettiin kutsua 0.9-versioksi erotteluna vasta myöhempien versioiden ilmestymisen jälkeen. Seuraaville HTTP-versioille 1.0 ja 1.1 sitten löytyykin jo tarkemmat kuvaukset [RFC1945, RFC2616].

HTTP/0.9-version kuvauksessa on esitetty, millainen kutsu selaimen on lähetettävä, jotta haluttu WWW-dokumentti saadaan siirrettyä. Ajatellaan tilannetta, jossa käyttäjä haluaa nähdä example.com-palvelimen juuridokumentin. Tilanne voisi syntyä käyttäjän kirjoitettua ”example.com”-osoitteen selaimen osoiteriviin tai siirryessä selaimellaan HTML-dokumentissa a-tunnisteella määriteltyn hyperlinkkiin:

```
<a href="http://example.com">linkki</a>.
```

Yhteyden avaamiseksi täytyy ensin löytää example.com -palvelimen IP-osoite, kuten luvussa 2.2 selvitettiin. Numeerisen osoitteen löydyttyä avataan siihen TCP-sovike (englanniksi ”socket”) oletuksena porttiin 80. Avatun yhteyden yli lähetetään teksti ”GET”, välilyönti ja sitten halutun dokumentin osoite. GET on pyyntötyypeistä yleisin ja yksinkertaisin. Se on ollut HTTP:ssä mukana jo ensimmäisestä 0.9-versiosta lähtien.

Polun lisäksi on mahdollista antaa myös toiminto [RFC1738]:

```
http://example.com/polku?toiminto.
```

Toimintomerkkijono välittyy palvelimelle HTTP-pyyntön mukana sinällään. Palvelimella ajettava dynaaminen ohjelma voi merkkijonon perusteella laatia sopivan vastauksen. Jos tarjottava palvelu olisi vaikkapa ruotsi-suomi -sanakirja, voisi toimintomerkkijono olla käännettävä sana ja vastaus sanan käännö.

Vastauksena palvelin lähettää pyydetyn dokumentin, joka on HTML-tunnistein merkattua tekstiä. HTML-dokumenttien lisäksi palvelin voi tarjota muitakin tiedostoja, vaikkapa kuvia tai äänitiedostoja. Koska tiedostot muodostavat usein kokonaisuuden, olisi tehokkaampaa lähettää toisiinsa liittyvät tiedostot kerralla, avaamatta joka kerta uutta yhteyttä. 1.1-versiosta lähtien saman yhteyden yli voidaanakin tehdä useampia pyyntöjä, avaamatta uutta sovitinta pyyntöjen välillä [Wikipedia:HTTP].

Tärkeä uudistus 1.1-versiossa oli myös mahdollistaa useamman verkkonimen palvelu saman IP-osoitteen takana, välittämällä polun lisäksi myös verkkonimi GET-pyyntöön yhteydessä ”Host:”-kentässä [RFC2616, 128]. Tämän ansiosta vaikkapa IP-osoite 1.2.3.4 voisi olla WWW-palvelin sekä osoitteille example.com että example.net ja palvelella verkkonimen perusteella eri HTML-dokumentin.

3.2 POST

Käyttäjä voi lähettää lomakkeen submit-painikkeella tai painamalla näppäimistön enter-näppäintä tekstikentässä. JavaScriptin submit() -metodi voi myös käynnistää lomakkeen lähettämisen, kuten listauksessa 3.1 esitetään.

Listaus 3.1: Lomakkeen lähettäminen JavaScriptistä

```
<html><body>
<form id=lomake>
  <input type=text>
</form>
<script>
  document.getElementById('lomake').submit();
</script>
</body></html>
```

Kun lomake lähetetään, käy selain läpi lomakekentät ja tekee niiden perusteella HTTP-pyyntöön palvelimelle. Lomakkeen tiedot täytyy siis jollakin tavalla sarjottaa, jotta ne voidaan lähettää. Sarjoitusmuoto riippuu HTML:n FORM-tagin ENCTYPE-tribuutista. Sen mahdolliset arvot ovat ”application/x-www-form-urlencoded” (oletusarvo), ”multipart/form-data” tai ”text/plain”. Kun palvelin saa lähetetyt tiedot, se saa selville käytetyn sarjoitustavan tutkimalla Content-Type -otsaketta (englanniksi ”header”) [HTML5Nightly, 4.10.22.3].

3.2.1 Enctype application/x-www-form-urlencoded

Jos enctype on "application/x-www-form-urlencoded", lähtevät lomakkeen tiedot URLeista tuttuun tapaan avain=arvo -pareina. Listauksessa 3.2 on esimerkki lomakkeesta, sekä sen lähettämisestä syntyvä HTTP-pyyntö, kun "eka" ja "toka"-kenttiin on kirjoitettu "aa", "bb".

Listaus 3.2: Lomakkeen sarjoittuminen

```
<form method=POST enctype="application/x-www-form-urlencoded">
<input type=text name=eka>
<input type=text name=toka>
<button type=submit>Submit</button>
</form>

POST /lomake-esimerkki HTTP/1.1
Host: example.com
eka=aa&toka=bb
```

3.2.2 Enctype multipart/form-data

Kun enctype on "multipart/form-data", on HTTP-pyyntöissä kentät eroteltu toisistaan valitsemalla erottimeksi merkkijono, jota ei muuten esiinny kutsussa. Tästä sarjoitustavasta on esimerkki listauksessa 3.3.

Listaus 3.3: Multipart-lomakkeen sarjoittuminen

```
<form method=POST enctype="multipart/form-data">
<input type=text name=eka>
<input type=text name=toka>
<button type=submit>Submit</button>
</form>

POST /lomake-esimerkki HTTP/1.1
Host: example.com
-----WebKitFormBoundary4SWARQTr0PaJvmqk
Content-Disposition: form-data; name="eka"

aa
-----WebKitFormBoundary4SWARQTr0PaJvmqk
Content-Disposition: form-data; name="toka"

bb
-----WebKitFormBoundary4SWARQTr0PaJvmqk --
```

3.2.3 Enctype text/plain

Tiedot siirtyvät avain=arvo -pareina rivivaihdoin erotettuna, kuten listauksesta 3.4 käy ilmi.

Listaus 3.4: Text/plain-lomakkeen sarjoittuminen

```
<form method=POST enctype="text/plain">
  <input type=text name=eka>
  <input type=text name=toka>
  <button type=submit>Submit</button>
</form>
```

```
POST /lomake-esimerkki HTTP/1.1
Host: example.com
eka=aa
toka=bb
```

”Text/plain”-ENCTYPE on tarkoitettu ihmisen luettavaksi, eikä sovellu todelliseen käyttöön palvelinohjelman kanssa, sillä jäsentämisen tulos ei välttämättä ole yksikäsitteinen [HTML51Nightly 4.10.22.8]. Esimerkki epäselvästä lomakkeesta ja sen lähettämisen synnyttävästä HTTP-kutsusta listauksessa 3.5. Esimerkissä ei olisi selvää kuuluisiko toka-muuttujan arvo olla ”bb” vai ”cc”.

Listaus 3.5: Epäselvä text/plain-lomakelähetys

```
<form method=POST enctype="text/plain">
<textarea name=eka>aa
toka=bb</textarea>
<input name=toka value=cc>
<button type=submit>Submit</button>
</form>
```

```
POST /lomake-esimerkki HTTP/1.1
Host: example.com
eka=aa
toka=bb
toka=cc
```

Edellä mainitut GET ja POST ovat yleisimmät WWW-selaimesta lähetettävät HTTP-pyyntö. Niitä kutsutaan myös verbeiksi (”verb”). Pyyntötyyppien määrä on muuttunut HTTP-versioiden mukana. Ensimmäisessä HTTP/0.9 -versiossa ainoa pyyntötyyppi oli GET [Mozilla HTTP]. Seuraavassa HTTP/1.0 -versiossa tulivat mukaan uusina päätyyppinä HEAD ja POST, sekä lisätyypit PUT, DELETE, LINK ja UNLINK [RFC1945].

Seuraavaksi ilmestyi vuonna 1997 versio HTTP/1.1 [RFC2068]. Siinä esiteltiin uutena tyyppinä PATCH. Myöhemmin standardista julkaistiin päivitetty versio, jossa juuri esitelty PATCH-pyyntötyyppi jälleen poistettiin. Lisäksi poistuiivat aiemmin mukana olleet LINK ja UNLINK. Poistojen syynä olivat niiden vähäinen käyttö [RFC2616, 174]. Kirjoitushetkellä uusin versio on edelleen vuonna 1997 ilmestynyt ja vuonna 1999 paranneltu HTTP/1.1. Versiossa tuli uutena pyyntötyyppinä mukaan CONNECT, jolla voidaan avata yhteystunneli välityspalvelimelle. Versio myös laajensi protokollaa lisäämällä MIME-tyyppiset viestit, sekä ottamalla mukaan metatietoa siirrettävästä tiedosta [RFC2616, 6]. HTTP:n seuraava tekeillä oleva versio on HTTP/2.0.

GET/POST -kutsujen suorittaminen onnistuu HTML-koodista linkkien ja lomakkeiden avulla, mutta muiden verbien käyttöön vaaditaan ohjelmointia. Seuraavassa kuvaillaan lyhyesti nykyisten pyyntötyyppien merkitys.

3.3 Muut pyyntötyypit

3.3.1 CONNECT

CONNECT-verbi on varattu tulevaan käyttöön vaihdettaessa välityspalvelimesta tunneliksi. WWW-välityspalvelimet toimivat välikätenä selaimen ja varsinaisen sisältöä tarjoilevan WWW-palvelimen välissä. Esimerkiksi työpaikalla välityspalvelin voi estää pääsyä tietyille verkkosivuille tai tallentaa sivuja välityspalvelimen välimuistiin, jotta pääsy niihin nopeutuisi.

Välityspalvelinten ongelmakohta on salatun tiedon siirto HTTPS-protokollalla. Jos välityspalvelin tekee HTTPS-kutsun WWW-palvelimelle ja välittää tiedon edelleen salaamattomana käyttäjälle, heikentyy tietoturva suoraan yhteyteen verrattuna. Tilanteen parantamiseksi esiteltiin [Luotonen 1998] CONNECT-pyyntötyyppi, jolla voidaan avata yhteystunneli. Tunnelia käytettäessä tieto välittyy edelleen välityspalvelimen kautta, mutta välityspalvelin ei tulkitse sisältöä millään lailla. Välityspalvelin ainoastaan välittää vastaanotetut ja lähetetyt tavut sinällään käyttäjän puolesta. HTTPS:n tapauksessa salattua tietoa ei tarvitse avata välityspalvelimella, eikä välityspalvelimen tarvitse edes ymmärtää salausta, vaan tietovirta välitetään heti sen saavuttua käyttäjälle sen sisältöön puuttumatta.

3.3.2 DELETE

DELETE-verbi pyytää palvelinta tuhoamaan mainitun dokumentin. Se on suunniteltu luontevammaksi vaihtoehdoksi resurssin tuhoamiseen POST-kutsun kaut-

ta. HTTP/0.9-versiossa ainoa tapa oli tuhoamisen pyytäminen POST-verbin kautta palvelinsovelmakohtaisella toteutuksella:

```
POST http://example.com/dokumentti/tuhoa.
```

HTTP/1.0-versiosta lähtien voitaisiin tuhoamista pyytää myös luontevammin DELETE-verbillä:

```
DELETE http://example.com/dokumentti.
```

3.3.3 PUT

PUT-verbiä käytetään tiedostojen lähettämiseen palvelimelle. PUT-kutsu lähettää palvelimelle uuden resurssin tiettyyn polkuun tai korvaa nykyisen. Vastauksen statuskoodista selviää, korvasiko resurssi olemassaolevan vai luotiinko uusi. Statuskoodi 201 viestittää uuden resurssin luomisesta, kun taas statuskoodit 200 tai 204 kertovat päivityksestä [RFC2068, 9.6].

Monen sovelluksen vaatimuksiin kuuluu CRUD-toiminnallisuutta (Create Read Update Delete), eli tietuiden luomista, lukemista, päivittämistä tai tuhoamista. GET-, DELETE- ja PUT-verbit tavoittelevat näiden yleisten toimintojen sisällyttämistä jo HTTP-metodeihin. GET-verbiä voidaan edelleen käyttää lukemiseen, kuten ennenkin. Luominen onnistuu PUT-verbillä silloin, kun URL viittaa uuteen resurssiin. Jos viitataan jo olemassaolevaan resurssiin, PUT-verbiä voidaan käyttää tiedon päivittämiseen [RFC1945, D.1.1]. Tuhoamisesta taas vastaa DELETE-verbi.

3.3.4 TRACE

TRACE-verbi on suunniteltu auttamaan verkkoylläpitäjää ongelmatilanteiden ratkaisemisessa. Kun palvelimelle lähetetään TRACE-verbinen HTTP-pyyntö, palvelin toistaa pyynnön HTTP-otsakkeet takaisin kutsujalle.

4 HTTP-KUTSUJEN VÄLITTYMINEN WWW-PALVELIMELTA PALVELINSOVELMALLE

Kun lomake lähetetään, on palvelimella oltava ohjelma, joka ottaa lähetetyt tiedot vastaan. WWW-palvelinohjelmat vastaavat staattisesta sisällöstä, kun taas dynaaminen WWW-palvelinohjelmointi tehdään yleensä jollakin komentosarjakielillä. Eräitä suosittuja komentosarjakieliä ovat PHP, Python ja Perl.

Tyypillisesti kutsujen vastaanottamisesta on vastuussa tehtävää varten laadittu WWW-palvelinohjelma, joka välittää kutsut palvelinsovelmille. Kutsujen vastaanottaminen suoraan komentosarjakielistäkin on mahdollista BSD-sovikkeiden API:n kautta. Komentosarjasta käsin voitaisiin luoda sovike `socket()`-funktioilla, yhdistää se `bind()`-funktioilla tiettyyn porttiin ja sitten odottaa kutsuja `listen()`-funktioilla [Wikipedia:Berkeley Sockets]. Tällöin kuitenkin menetettäisiin WWW-palvelinohjelmien edut. WWW-palvelinohjelmaa käytetään välikätenä, sillä sen ansiosta palvelinsovelmia voi olla samassa portissa useampia. Järjestelyn ansiosta palvelin voi valvoa sovelmien käyttämää tiedonsiirtomäärää siten, että yksittäinen ohjelma ei vie koko kaistanleveyttä. Palvelin voi dynaamisten komentosarjojen ajamisen lisäksi tarjoilla myös staattista sisältöä sekä pitää lokia saapuvista kutsuista. Sopiva ajettava palvelinsovelma voidaan valita polun tai isäntänimen perusteella. Isäntänimeen perusteella palvelevaa palvelinta kutsutaan virtuaalipalvelimeksi (englanniksi ”virtual hosting”). Esimerkkejä suosituista WWW-palvelinohjelmista ovat Apache, Nginx ja lighttpd.

Jos palvelinohjelma ottaa lomakekutsun vastaan, mutta palvelinsovelman ohjelmalogiikka on toteutettu WWW-palvelinohjelman ulkopuolella, on kutsun käsittelemiseksi tiedot jollakin tavoin välitettävä sovelmalle. Komentosarjakielen tulkki tai käännetty ohjelma on siis käynnistettävä (CGI), tai sen on oltava jo valmiiksi käynnissä (FastCGI). Komentosarjakielen tulkki voi olla myös sisällytetty osaksi palvelinohjelmaa (esim. `mod_php` ja `mod_perl`). Koska lähestymistapoja on erilaisia, ei tietojen välittämiseen ole vain yhtä standardia, vaan WWW-palvelimen asetuksista riippuen on monta mahdollista tapaa välittää HTTP-kutsu palvelinsovelmalle. Tarkastellaan seuraavaksi joitakin yleisesti käytössä olevia tapoja.

4.1 Palvelinsovelman käynnistäminen yksittäisen kutsun palvelemiseksi CGI:llä

CGI (Common Gateway Interface) on standardi HTTP-pyyntöjen välittämiseen palvelinsovelmalle, joka on käynnistetty varta vasten yhden kutsun käsittelyä varten. Siinä HTTP-kutsun tiedot välitetään ympäristömuuttujien kautta. Ratkaisuun päädyttiin vuonna 1993 `www-talk` -nimisellä sähköpostilistalla [Wikipedia:Common Gateway Interface]. Kun WWW-palvelin saa kutsun, se valitsee kutsun polun ja asetuksiensa perusteella sopivan CGI-ohjelman, muuntaa jäsennetyn HTTP-kutsun CGI-kutsuksi, ajaa komentosarjan ja sitten muuntaa CGI-vastauksen HTTP-vastaukseksi [RFC3875]. Palvelinsovelma sulkeutuu kutsun palvelun jälkeen.

Jos kutsu on POST-tyyppinen, välitetään lomakkeelle syötetyt tiedot CGI-komentosarjalle standardin syöttöväylän (`stdin` - standard input) kautta. GET-verbin tapauksessa tiedot välittyvät ympäristömuuttujan `QUERY_STRING` kautta. GET- tai POST -sisällön lisäksi CGI-komentosarja saa tietoa itse kutsusta, kutsujasta ja kutsun palvelleesta WWW-palvelimesta. Esimerkiksi ympäristömuuttujassa `REMOTE_ADDR` välitetään kutsun lähettäjän IP-osoite. Muuttujasta `REQUEST_METHOD` puolestaan ilmenee pyyntötyypin verbi kuten ”GET” tai ”POST”.

CGI on ratkaisuna varsin yksinkertainen mutta hidas, sillä palvelinsovelma käynnistetään uudelleen joka kutsun palvelemista varten. Useille nettisivuille se on täysin riittävä ratkaisu, mutta jos palvelinresursseja halutaan säästää tai kutsujen palvelemisen viivettä vähentää, olisi parempi jos ohjelma (kuten komentosarjakielen tulkki) voisi olla jo valmiiksi käynnissä.

4.2 Kutsun välittäminen jo käynnissä olevalle palvelinsovelmalle FastCGI:llä

Koska tulkkiprosessin luominen ja sen tuhoaminen jokaista kutsua varten ei ole tehokasta, FastCGI-protokolla kehitettiin ratkaisuksi tulkin pitämiseksi käynnissä kutsujen välillä. Yksi tai useampia palvelinsovelmia voi siis olla jo valmiiksi käynnissä odottamassa saapuvia kutsuja, joten kutsun saavuttua käynnistämiseen ei kulu aikaa. Koska prosessi pysyy käynnissä, se voi myös pitää usein tarvittavia tietoja muistissa tai tietokantayhteyksiä avoinna, joten aikaa säästyy edelleen.

FastCGI-protokollan kehitystyön teki Open Market -yritys vuonna 1996 [FastCGI] omaa ”Open Market Web Server” WWW-palvelintaan varten, mutta ny-

kyään kaikki suosituimmat WWW-palvelinohjelmat tukevat sitä. Palvelinsovelma käynnistetään ja jää odottamaan sovikkeen kautta tulevia kutsuja. FastCGI ei saa kutsun tietoja ympäristömuuttujissa tai standardin syöttöväylän kautta, vaan sovikkeiden kautta. Tämä mahdollistaa myös sen, että esikäynnistettyjen selainsovelmien ei tarvitse välttämättä olla samalla palvelimella WWW-palvelinohjelman kanssa. WWW-palvelin vastaa sovikkeen luomisesta, välittäen sen FastCGI-prosessille avoimen tiedostokuvaaajan kautta prosessia käynnistettäessä [FastCGI 2.2].

FastCGI-ohjelma voi saman sovikeyhteyden kautta saada useitakin pyyntöjä WWW-palvelimelta. Protokollassa WWW-palvelimelta lähetetään sovikkeen yli palvelinsovelmalle FastCGI-tietueita, joita on muutamia eri tyyppisiä. Tietueet sisältävät vaaditut avain-arvoparit käyttäjältä tulleilta HTTP-kutsuilta, mutta lisäksi WWW-palvelin voi saada palvelinsovelmalta metatietoja, kuten kuinka monta yhtäaikaista kutsua se kykenee palvelemaan [FastCGI 5].

4.3 SCGI

SCGI (”Simple Common Gateway Interface alternative”) on tarkoitettu yksinkertaisemmaksi toteuttaa kuin FastCGI. Se ei määrittele miten WWW-palvelin löytää SCGI-komentosarjan, vaan ainoastaan miten kutsun tiedot välittyvät SCGI-ohjelmalle. SCGI-standardi ei edes määrittele vastauksen muotoa [SCGI] tai millä tavoin kutsun tiedot välitetään SCGI-ohjelmalle. Ainoastaan kutsun siirtoformaatti on määritelty.

Siirrossa SCGI-sovelmalle lähetetään sarja kutsuotsakkeita. CGI-ympäristömuuttujat välitetään sovelmalle muiden muuttujien joukossa, joten SCGI:n toteuttaminen vaatii tietoa myös CGI:n toiminnasta otsakkeiden suhteen. Ensimmäisen otsakkeen tulee aina olla `CONTENT_LENGTH`, joka kertoo koko sisällön koon. Kutsu välitetään jonona avain-arvopareja, joissa sekä avaimen ja arvon jälkeen tulee nollatavu. Avaimet ja arvot välittyvät netstringeinä. Otsakkeiden jälkeen lähetetään mahdollinen POST-sisältö. Otsakkeiden ja sisällön lisäksi mitään muuta ei lähetetä.

4.3.1 Netstring

Merkkijonojen lähettämiseen SCGI:ssä käytetään netstringeiksi kutsuttua tapaa [Netstrings], jossa merkkijonon koko välitetään ensin ASCII-numeroiksi koodattuna, jonka jälkeen tulee itse merkkijono ja sitten pilkku:

```
13:Hei, maailma!,,
```

Pilkkua ei lasketa mukaan merkkijonon kokoon. Netstringien etuna on, että toisin kuin hipsuilla tai lainausmerkeillä ympäröidyissä merkkijonoissa, ei netstring-merkkijonoissa esiinny mitään poikkeavasti käsiteltäviä merkkejä. Esimerkiksi jos merkkijono "O'Connels" haluttaisiin esittää hipsuin ympäröitynä, täytyisi siihen sisältyvä hipsu esittää muodossa 'O\'Connels'. Netstringeissä tällaista tarvetta ei tule. Koska merkkijonon koko tiedetään etukäteen, voi vastaanottaja myös ennakolta varata sille tarvittavan muistitilan.

4.4 Kutsun välittäminen Python-palvelinsovelmalle WSGI:llä

WSGI ("Python Web Server Gateway Interface") on tapa, jolla Python-ohjelma voi vastaanottaa kutsuja WWW-palvelimilta. CGI ja FastCGI eivät riipu palvelinsovelman ohjelmointikielestä, kun taas WSGI toimii vain Pythonin kanssa. Useille protokollille on saatavilla WSGI-kääre, joka tulee WWW-palvelimen ja WSGI:tä tukevan Python-ohjelman väliin. Näin WSGI-ohjelmia voidaan käyttää myös sellaisten WWW-palvelimien kanssa, jotka eivät suoraan tue WSGI:tä. Myös FastCGI:lle on tällainen kääre (englanniksi "wrapper") [fcgiapp]. Fcgiapp on Python-kääre FastCGI-protokollalle. Se siis mahdollistaa WSGI:n käytön "FastCGI:n yli" [fcgiapp].

Palvelimille on saatavilla myös lisäosia, joilla palvelimen saa tukemaan WSGI:tä suoraan. Apachelle tällainen lisäosa on `mod_wsgi` ja Nginxille "Nginx WSGI module". Lighttpd:lle suositeltu tapa on käyttää WSGI:tä FastCGI:n yli [Howto WSGI]. Palvelinpuolen Python-sovelma voisi tietenkin myös toteuttaa FastCGI-protokollan suoraan, mutta WSGI:n käyttö on huomattavasti yksinkertaisempaa. Lisäetuna sovelma voidaan myöhemmin siirtää käyttämään muitakin WWW-palvelimen ja sovelman välisiä kutsuntatapoja, kuten toimimaan CGI-komentosarjana.

WSGI-liitynnässä on kaksi puolta: palvelin ja sovelma. Kun palvelin haluaa välittää kutsun sovelmalle, palvelin kutsuu Python-koodissa määriteltäviä oliota, joka voi olla luokka tai funktio [PEP3333, Specification Overview].

WSGI ei määrittele, miten palvelin saa instanssin sovelmasta. Kun instanssi on kuitenkin jollakin palvelinkohtaisella tavalla saatu, sitä kutsutaan kahdella argumentilla:

```
result = application(environ, start_response).
```

Ensimmäisenä parametrina saadaan tietoa kutsusta. Environ-parametri sisältää sekä ympäristömuuttujat, että WSGI-spesifisiä tietoja. Toisena parametri-

na välitetään vastausfunktio. WSGI:n spesifikaatio antaa lyhyimmäksi WSGI-ohjelmaksi listauksen 4.1 kaltaisen esimerkin.

Listaus 4.1: Minimalistinen WSGI-esimerkki

```
def simple_app(environ, start_response):
    status = '200 OK'
    response_headers = [('Content-type', 'text/plain')]
    start_response(status, response_headers)
    return ['Hei, maailma']
```

Sovelma välittää `start_response`-funktion avulla tietoja luomastaan vastauksesta. Ensimmäisenä argumenttina kutsuvalle WWW-palvelimelle vastauksen statuskoodin. Toisena argumenttina sovelma lähettää listan HTTP-otsakkeita avainarvo -pareittain. Siinä voitaisiin palauttaa esimerkiksi sisällön tyyppin kertova Content-Type -otsake tai evästeet asettava Set-Cookie -otsake. Kutsun pääasiallinen sisältö palautetaan kutsuttavan olion paluuarvona. HTML:ää tuottavan sovelman tapauksessa paluuarvo voisi siis olla HTML-koodia tai kuvia tuottavassa sovelmassa kuvadataa.

WSGI on ideana siis varsin yksinkertainen. WSGI onkin suunniteltu mahdollisimman helpoksi toteuttaa, jotta WWW-sovellusten kehittäjät ottaisivat sen käyttöön [PEP3333, Rationale and Goals]. Sen käyttö saattaa olla myös helpompaa kuin FastCGI:n lisääminen omaan sovellukseen, mikäli sopivalla kääreellä tai palvelimen lisäosalla sitä voidaan tukea.

4.5 Sovelman sisällyttäminen WWW-palvelinohjelmaan

Vaihtoehtona FastCGI:lle eräs tapa välttää palvelinsovelman käynnistämisestä aiheutuvaa viivettä on sisällyttää palvelinsovelma palvelinprosessiin lisäosana. Tällöin ei tarvita prosessien välistä kommunikointia, koska sekä palvelinsovelma että WWW-palvelin ovat samaa prosessia. Tämä saattaa olla hyvä ratkaisu silloin, kun palvelinsovelma on samalla palvelimella WWW-palvelinohjelman kanssa. WWW-palvelimeen voidaan sisällyttää komentosarjakielen tulkki tai muita haluttuja toimintoja.

Lisäosan käyttö vaatii, että käytetylle WWW-palvelimelle on saatavilla juuri halutun komentosarjakielen tulkki. Apache-palvelimelle näitä on saatavilla suuri kirjo. Joitakin tarjolla olevia komentosarjakielten tulkkeja Apachelle ovat esimerkiksi `mod_php`, `mod_python` ja `mod_perl`. Nginx-palvelimelle on saatavilla ainakin Lua- ja JavaScript-lisäosat [Nginx third party modules].

Lighttpd-palvelimelle tulkkeja on saatavilla heikommin, lähinnä vain Lua-

kielen `mod_magnet`. `Lighttpd` on suunniteltu enemmänkin CGI:n, SCGI:n tai `FastCGI`:n kanssa käytettäväksi [Lighttpd luku 10].

4.6 Kutsun tulkinta palvelinsovelmassa

Kun WWW-palvelin on käynnistänyt sovelmakielen tulkin ja välittänyt sille kutsun tiedot, on sovelman seuraavaksi laadittava kutsun tietojen perusteella vastaus. Kun on kyse POST-tyyppisestä kutsusta, löytyvät lähetetyt tiedot HTTP-kutsun body-osasta. PHP (rekursiivinen lyhenne sanoista ”PHP: Hypertext Preprocessor”) on hyvin suosittu Rasmus Lerdorfin vuonna 1995 kehittämä WWW-kehitykseen suunniteltu ohjelmointikieli. Koska kieli on juurikin suunniteltu dynaamisten WWW-sivujen laatimiseen, on sillä helppoa tulkita lomakelähetykset. Listauksessa 4.2 on esimerkki, joka tulostaa käyttäjän kirjoittaman viestikentän sisällön.

Listaus 4.2: PHP-esimerkki

```
<?php
if ($_POST) {
    echo htmlspecialchars($_POST['viesti']);
} else { ?>
    <form method=POST>
        <input type=text name=viesti>
        <button type=submit>Submit</button>
    </form>
<?php } ?>
```

PHP jäsentää saapuvan kutsun ja esittää ne kehittäjälle `$_POST`-hakurakenteena. Sekä `ENCTYPE` ”application/x-www-form-urlencoded” että ”multipart/form-data” toimivat muutoksitta.

Esitetään toisena esimerkkinä kutsun tulkinta Python-ohjelmasta. Kun POST-kutsu saadaan WSGI:n kautta, löytyy `environ`-sanakirjan ”wsgi.input”-attribuutista tiedostomainen tietovirta, jonka kautta sisältö voidaan lukea [PEP3333, `environ Variables`]. Kutsun jäsenyykseen soveltuva funktio löytyy `cgi`-modulista. `Cgi`-modulin `parse`-funktio osaa tulkita sarjoitustavan ympäristön `Content-Type` -otsakkeen perusteella. Listauksessa 4.3 WSGI-esimerkkiohjelma, joka tulostaa sille lähetetyt lomaketiedot.

Listaus 4.3: Python WSGI-esimerkkiohjelman

```
def app(environ, start_response):
    response_headers = [('Content-type', 'text/plain')]
    start_response('200 OK', response_headers)
    import cgi
    out = cgi.parse(environ['wsgi.input'], environ)
    return ['Lomakkeen tiedot: %s' % out]
```

5 TIEDONSIIRTOMUOTOJA

Perinteisin tapa rakentaa dynaamisia WWW-sovelluksia oli laatia koko HTML-vastaus palvelimella GET- ja POST-tyyppisten HTTP-pyyntöjen vastauksena. Esimerkiksi Tampereen Yliopiston Informaatiotutkimuksen Laitoksen (nykyisin Informaatiotutkimuksen ja Interaktiivisen Median TRIM-tutkimuskeskus) WWW-pohjainen tiedonhaun opetus- ja tutkimusohjelma Query Performance Analyzer [Sormunen and Pennanen, 2004] perustuu tähän.

Myöhemmin uusien kommunikointikanavien myötä syntyi mahdollisuus vähentää käyttäjän kokemaa viivettä, siirtämällä pienempiä tietopalasia kokonaisen HTML-vastauksen lähettämisen sijaan. Esitellään muutamia suosittuja tiedonsiirtomuotoja, jotka erityisesti soveltuvat rakenteisen tiedon siirtoon näiden viestintäkanavien yli.

5.1 JSON-tiedonsiirtomuoto

JSON on lyhenne sanoista ”JavaScript Object Notation” (JavaScript-olionotaatio). Vaikka lyhenne viittaaakin JavaScript-kieleen, on kyseessä kuitenkin ohjelmointikielistä riippumaton yksinkertainen tiedonsiirtomuoto [JSON.org]. JSON on saavuttanut vähitellen suuren suosion. Viittaus JavaScript-kieleen tulee siitä, että tiedot siirtyvät aivan kuten oltaisiin siirtämässä JavaScript-kielistä ohjelmakoodia, jossa kyseinen tietorakenne luotaisiin.

Listaus 5.1: JSON-esimerkki

```
{
  "Merkurius" : {
    "kiertoaika" : 88,
    "kiertolaiset" : []
  },
  "Venus" : {
    "kiertoaika" : 225,
    "kiertolaiset" : []
  },
  ...
}
```

JSON on muihin tiedonsiirtomuotoihin verrattuna helppo ymmärtää ja lukea. Sitä on myös helpompi jäsentää tai luoda ohjelmallisesti. Se on SOAP-tiedonsiirtomuotoon verrattuna varsinkin sisennettynä luontevampaa lukea, koska itse tietosisällön lisäksi ei formaatti vaadi muuta ympärilleen. JSON tarjo-

aa minimaalisen joukon sääntöjä, joilla pelkistetty tiedonsiirto on mahdollista [RFC4627].

Jos haluttaisiin siirtää esimerkiksi aurinkokuntaa kuvaava taulukko, voisi sen alkuosa muistuttaa JSON-tiedonsiirtomuodossa listauksen 5.1 esimerkkiä. Vertailuksi sama sisältö SOAP-muotoisena ("Simple Object Access Protocol") listauksessa 5.2 [W3 SOAP]. SOAP on XML-merkintäkieleen perustuva sarjoitusmuoto, jota käytetään samankaltaisiin tehtäviin kuten JSON-muotoakin.

Listaus 5.2: Esimerkki SOAP-muotoisesta sisällöstä

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://example.com/envelopes/aurinkokunta/"
  SOAP-ENV:encodingStyle="http://example.com/encodings/
  aurinkokunta/" />
  <SOAP-ENV:Body>
    <m:AurinkokuntaResponse xmlns:m="Some-URI">
      <Planeetta>
        <Nimi>Merkurius</Nimi>
        <Kiertoaika>88</Kiertoaika>
        <Kiertolaiset></Kiertolaiset>
      </Planeetta>
      <Planeetta>
        <Nimi>Venus</Nimi>
        <Kiertoaika>225</Kiertoaika>
        <Kiertolaiset></Kiertolaiset>
      </Planeetta>
      ...
    </m:AurinkokuntaResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

JSON ei tarjoa tietojen vastaanottajalle kenttänimien lisäksi minkäänlaista kuvausta tietosisällöstä, joten lähettäjän ja vastaanottajan on ennalta tiedettävä datan merkitys (sama koskee myös luvun 5.3 protokollabuffereita). Moniin käytännön tilanteisiin tämä on riittävää, mutta mikäli laajempaan palvelukuvaukseen on tarvetta, on siihen kehitetty JSON-WSP -protokolla. Protokollan avulla on mahdollista antaa lisätietoja metodeista ja tietotyypeistä [Wikipedia:JSON-WSP].

5.2 JSONin jäsentäminen valmiilla kirjastoilla

Tiedonsiirtomuotona JSON on yksinkertainen myös vastaanottajan kannalta. XML-jäsentäjän kirjoittaminen on hankala tehtävä, mutta JSON-parsija on mahdollista toteuttaa vähemmällä työllä. Todellisissa jäsentäjätoteutuksissa on koodirivejä esimerkiksi ActionScriptillä 327 riviä [as3corelib], Lispillä 730 riviä [cl-json] ja Pythonilla 389 riviä [simplejson]. Pythonin versiosta 2.6 lähtien on JSON-jäsentäjä myös sisäänrakennettuna.

Koska JSON on jo valmiiksi javascriptiä, on yksinkertainen javascript-toteutus vain yksi rivi:

```
eval('(' + merkkijono + ')').
```

Tämä ei kuitenkaan ole paras mahdollinen toteutus, sillä monessa tilanteessa JSON-muotoisen tiedon sisältävä merkkijono voisi tulla käyttäjältä. Sen sisällön turvallisuuteen ei silloin siis voi luottaa, eikä välttämättä ole aina turvallista antaa eval-funktion vapaasti ajaa käyttäjän ohjelmakoodia.

Käytännössä kuitenkin JavaScript estää eri palvelinten ("origin") väliset HTTP-kutsut, joten JSON tulee luotetulta omalta palvelimelta. Luotettu oma palvelin tarkoittaa tässä resurssia, jonka URLissa on sama skeema, omistaja ja portti [HTML5Nightly].

Mikäli oma palvelin lähettäisi vaarallisia JSON-vastauksia, niin silloinhan myös alkuperäinen palvelimelta saatu JavaScript-koodi olisi jo valmiiksi sisällöltään kyseenalainen [Crockford, s. 139]. Jos JSON-muotoinen merkkijono on koostettu oikein, ei ongelmia synny. Jos palvelin sisällyttää luotuun JSONiin käyttäjältä saatuja merkkijonoja laatimatta JSON-vastausta oikein, voisi vaaratilanne kuitenkin periaatteessa syntyä. Varmuuden vuoksi voi myös JavaScriptissä käyttää evalin sijaan JSON-jäsentäjää.

5.3 Protokollabufferit

Protokollabufferit (englanniksi "Protocol Buffers") on laajennettava tiedonsiirtomuoto jäsenneen tiedon siirtämiseksi. Google käyttää tätä formaattia sisäisissä sovelluksissaan [protobuf]. Protokollabuffereilla voidaan esittää tietorakenteita kuten XML-muodossakin, mutta se on nopeampi jäsentää ja vaatii vähemmän tilaa. Haittapuolena tiiviimpää binäärimuotoa on ihmisen vaikeampi lukea, joka saattaa vaikeuttaa siirtomuodon ymmärtämistä ja virhetilanteiden selvittämistä.

Googlen dokumentaatioissa verrataan protokollabuffereita tiedon esittämiseen XML-muodossa [Protocol Buffers Developer Guide]. Protokollabuffereiden eduksi mainitaan seuraavaa:

- Yksinkertaisuus
- 3-10 kertaa pienempi koko
- 20-100 kertaa nopeampi käsittelyaika
- Yksiselitteisyys
- Kääntäjän luomat luokat helppoja käyttää ohjelmakoodista

Haittapuolina mainitaan protokollabuffereiden sopimattomuus HTML-tyylisen merkintäkielen esitystapana, sillä rakenteiden esittäminen tekstin seassa ei onnistu luonnekaasti. XML-esitystavalla esimerkiksi otsikon merkitseminen muun tekstin seassa <H1>-elementillä onnistuu helpommin.

Protokollabuffereita käytetään huomalla tietoa kuvaileva .proto -tiedosto. Tiedoston perusteella luodaan kääntäjällä eri ohjelmointikielille sopivat luokat. Kirjoitushetkellä Googlen projekti [protobuf] tukee suoraan Java-, C++- ja Python-ohjelmointikieliä. Muillekin ohjelmointikielille on saatavilla kirjastoja. Esimerkiksi JavaScript-kielisestä selainohjelmasta voidaan viestiä protokollabuffereilla palvelimelle protojs-kirjaston avulla [protojs].

Listaus 5.3: Esimerkki .proto-tiedostosta

```
message Aurinkokunta {
  repeated Planeetta planeetta = 1;
}

message Planeetta {
  required string nimi = 1;
  required int32 kiertoaika = 2;
  repeated string kiertolaiset = 3;
}
```

Listauksessa 5.3 on esimerkki .proto -tiedostosta (esimerkki.proto). Siitä saadaan komentorivillä protoc-kääntäjällä java-luokka *Esimerkki.java* seuraavasti:

```
protoc esimerkki.proto --java_out=..
```

Generoitu Java-tiedosto määrittelee luokkia, joita käyttäen ohjelmoija voi omassa ohjelmakoodissaan luoda protokollabufferiformaatin mukaisia viestejä [Protocol Buffer Basics]. Listauksessa 5.4 luodaan viesti, joka on osittainen kuvaus aurinkokunnasta.

Listaus 5.4: Generoitujen luokkien käyttö

```
Aurinkokunta aurinkokunta =
    Aurinkokunta.newBuilder()
        .addPlaneetta(
            Planeetta.newBuilder()
                .setNimi("Mars")
                .setKiertoaika(687)
                .addKiertolaiset("Phobos")
                .addKiertolaiset("Deimos")
            )
        .build();

try {
    OutputStream os = new FileOutputStream("viesti");
    aurinkokunta.writeTo(os);
} catch (Exception e) { }
```

Ohjelmalistaus tallettaa tiedostoon protokollabufferiformaatin mukaisen viestin. Koska formaatti on binäärimuotoinen, eikä ihmisen luettavaksi suunniteltu, siitä ei ole hyödyllistä näyttää tässä esimerkkiä. Huomattavaa on kuitenkin sen lyhyt koko, vain 27 tavua.

5.4 Tiedostojen lähettäminen lomakkeella

Jos WWW-sivulle halutaan lisätä mahdollisuus tiedostojen lähettämiseen, voidaan se HTML-koodissa tehdä ”file”-tyyppisen input-kentän avulla. Jos esimerkiksi haluttaisiin antaa käyttäjille mahdollisuus asettaa itselleen profiilikuva, voisi kuvan lähettämiseen käytettävä HTML-koodi olla kuten listauksessa 5.5.

Listaus 5.5: Python WSGI-esimerkkiohjelma

```
<form method=POST action=upload>
    <input type=file name=profiilikuva>
    <input type=submit>
</form>
```

Käyttäjän painaessa esimerkkilomakkeen submit-nappulaa avaa selain yhteyden lomakkeen action-attribuutin määrittelemään palvelinpolkuun ja tekee

POST-tyyppisen pyynnön. Lähetys voidaan tulkita palvelimella edelleen samalla tavalla, kuten aikaisemman luvun listauksen 4.3 Python-esimerkissä näytettiin. Lähetetyn tiedoston binäärisisältö tulee saataville kyseisessä esimerkissä *out*-sanakirjaan.

Alkuperäisessä RFC-dokumentissa [RFC1867] vain asiaa mainitsematta oletetaan, että tiedostoa siirtäessä käytettäisiin ”multipart/form-data”-ENCTYPEä. Vasta myöhemmin HTML4-standardissa todetaan suoraan, että ”multipart/form-data” ENCTYPEä tulee käyttää (”should be used”) [HTML4.01 forms]. RFC-dokumentti ei mainitse, mitä tapahtuu jos tiedostoa yritetään siirtää ”application/x-www-form-urlencoded”-ENCTYPEllä, mutta HTML-spesifikaatiosta [HTML5.1 4.10.22.6] ja kokeellisesti huomataan, että tällöin siirtyy tiedoston sisällön sijasta ainoastaan sen nimi.

Käytännössä pelkän input-kentän sijaan käytetään usein yhdistelmäratkaisua, jossa käyttäjälle näytetään Flash- tai Java-pohjainen kuvanlähetystyökalu silloin, kun kyseiset selaimen lisäosat ovat saatavilla. Lisäosilla saavutetaan useissa tapauksissa käyttäjäystävällisempi lähetyskokemus. Niiden etuja ovat useamman tiedoston lähettäminen kerralla, sekä kuvien lähetyksen edistyksen seuraaminen edistymispalkin avulla. Varsinkin vanhemmilla selaimilla nämä ominaisuudet eivät ilman lisäosia ole mahdollisia.

5.5 Muita tiedonsiirtomuotoja

Tässä esitettyjen formaattien lisäksi saattaa joissakin tilanteissa olla suoraviivaisinta siirtää rakenteisen tiedon sijaan vain yksittäinen kenttä. Jos selain pyytää palvelinta suorittamaan jonkin toiminnon, voi vastauksena riittää pelkkä ”OK” ASCII-tekstinä osoittamaan toiminnon onnistunutta suoritusta.

Suosittu vaihtoehto on myöskin valmiin HTML-palasen lähettäminen palvelimelta. Sen sijaan, että lähetettäisiin rakenteinen viesti, jonka perusteella selain luo HTML-palasen ja esittää sen käyttäjän selaimessa, voidaan palanen luoda palvelimella. Tällöin selaimen tehtäväksi jää ainoastaan sen esittäminen. Vaikkakin tämä on usein helppo askel kokonaisten HTML-dokumenttien siirtämisestä viiveettömämpään käyttäjäkokemukseen, on rakenteisempien vastausten etuna HTMLlään perustumattomien asiakasohjelmien teon helpottuminen. Jos halutaan myöhemmin tehdä natiivi mobiilisovellus Android- tai iOS-alustoille, tekeminen helpottuu, jos palvelimen vastaukset ovat HTML-palasten sijaan jossakin helposti jäsennettävässä muodossa esitettyä rakenteista tietoa.

6 COMET-YHTEYDET JAVASCRIPT-SOVELMISTA

JavaScript-ohjelmointikielen kehitti Brendan Eich vuonna 1995 [InfoWorld]. Se on suosittu erityisesti, koska se on sisällytetty valmiina WWW-selaimiin. Netscape 2.0 oli ensimmäinen selain, jossa JavaScript-tuki oli mukana.

JavaScript nimenä saattaa aiheuttaa sekaannusta, koska sillä ei ole juurikaan yhteyttä Java-kieleen. Sen standardinkaan nimi ei ole ”JavaScript”, vaan ”ECMAScript” [ECMA-262]. ECMAScript-standardi poikkeaa nimenä JavaScriptin nimestä tavaramerkkikiistan vuoksi [InfoWorld]. Asiayhteys Javaan jää myös ainoastaan markkinointitasolle, sillä JavaScriptiä ajateltiin aluksi Javan kanssa käytettäväksi kevyemmäksi komentosarjakieleksi.

JavaScript suunniteltiin ensisijaisesti helpoksi tavaksi lisätä hieman dynaamista toiminnallisuutta WWW-sivuihin [InfoWorld]. JavaScriptin ajaminen palvelinpuolen kielenäkin on ollut pitkään mahdollista. Ensimmäiset toteutukset ilmesivät jo vuonna 1994. Nykyään käyttö palvelinsovelmakielenä on huomattavasti kasvanut Node.js-alustan ansiosta. Node.js sisältää myös WWW-palvelimen. Eräs etu JavaScriptin käytölle palvelimella on saman ohjelmakoodin hyödyntäminen sekä palvelin- että asiakassovelmassa.

Alex Russell esitteli ”comet”-termin blogipostauksessaan vuonna 2006 [Comet: Low Latency Data for the Browser]. Selainsovelmalle yksinkertaisin tapa pysyä perillä palvelimen reaaliajassa muuttuvan tiedon päivityksistä on kysellä selaimelta tiedon nykytilasta uudelleen ja uudelleen. Tällaista nopeaan tahtiin tapahtuvaa kyselyä kutsutaan kiertokyselyksi (englanniksi ”polling”). Comet on yleisnimitys tekniikoille, joissa palvelin voi lähettää selaimelle viestejä ilman, että selain tarvitsisi jatkuvaa kiertokyselyä uusien päivityksien vastaanottamiseksi. Esittelyhetkellä ”Ajax”-termi oli jo yleistynyt tarkoittamaan sivulatausten välillä JavaScriptin avulla selaimesta palvelimelle tehtäviä pyyntöjä. Ajax on suositun puhdistusaineen nimi, joten myös comet-tekniikalle valittiin nimi toisen Comet-puhdistusaineen perusteella. Termin esittelyssä Russell kertoi comet-tekniikoille olevan yhteistä se, että ne käyttävät pitkäaikaista yhteyttä palvelimeen. Ajaxiin verrattuna comet-mallissa palvelin pääsee tekemään aloitteen tiedon siirtämiseksi, joten muuttunut tieto saadaan vähemmällä viiveellä.

6.1 Kiertokysely XMLHttpRequest-oliolla

Ajaxin jatkuvan kiertokyselyn hyvä puoli oli, että ratkaisu on ohjelmoijan kannalta helppo ymmärtää ja toteuttaa. Yksinkertaisin tapa siirtää tietoa jatkuva-

na virtana palvelimelta selaimen on laatia JavaScript-komentosarja, joka kysyy kiertokyselynä toistuvasti palvelimelta uusimpia tietoja. Komentosarja voi kysyä tietoja esimerkiksi kerran sekunnissa, kuten listauksessa 6.1 esitetään. JavaScriptistä voidaan tehdä HTTP-kutsuja XMLHttpRequest-oliolla. Vaikkakin olion nimessä on lyhenne ”XML”, ei olion käyttö kuitenkaan rajoitu vain XML-tiedon siirtoon. Nimi on historiallinen jäännös, sillä olio lisättiin alunperin IE-selaimen MSXML-kirjaston mukana.

Listaus 6.1: Kiertokysely XMLHttpRequest-oliolla

```
var r = new XMLHttpRequest();
setInterval(function () {
    r.open('GET', '/resurssi', true);
    r.onload = function () {
        console.log("Uusin tieto on: " + r.responseText);
    };
    r.send();
}, 1000);
```

Resursseja tällainen toteutus kuitenkin tuhlaa, sillä selaimen on tehtävä HTTP-pyyntöjä palvelimelle jatkuvasti, vaikka uutta tietoa ei olisikaan saatavilla. HTTP-pyyntöissä lähetetään aina HTTP-otsake uudelleen, joka saattaa sisältää paljonkin dataa, kuten evästeitä. Jatkuva kiertokysely siis tuhlaa myös tietoliikennekaistaa. Tekniikasta aiheutuu myös viivettä, sillä jos selain esimerkiksi sekunnin välein kysyy palvelimelta uusimman tiedon, olisi uutta tietoa saattanut olla saatavilla jo kyselyiden välissä. Toisaalta tietoa saatetaan kysyä myös turhaan, vaikka se ei olisi palvelimella muuttunut lainkaan edellisestä kyselykerrasta. Olisi parempi, jos uusi tieto saataisiin heti sen tultua saataville. Olisi myös tehokkaampaa, jos voitaisiin välttää uuden tiedon kysyminen silloin, kun sitä ei ole saatavilla.

6.2 Pitkäkiertokysely

Turhien kiertokyselyjen vähentämiseksi on olemassa hieman erilainen pitkäkiertokyselyksi (englanniksi ”long polling”) kutsuttu lähestymistapa. Pitkäkiertokysely parantaa tilannetta tavalliseen kiertokyselyyn verrattuna. Tavanomaisessa kiertokyselyssä palvelin välittömästi vastaa saamaansa HTTP-kutsuun, oli uutta tietoa saatavilla tai ei. Pidempikestoinen kiertokysely taas perustuu siihen, että selain jättää yhteyden auki vastaten kutsuun vasta sitten, kun uutta tietoa on tullut saataville. Sen sijaan, että palvelin heti vastaisi pyyntöön, se jääkin odottamaan kunnes uutta muuttunutta tietoa on saatavilla. Tämän ansiosta turhia kutsuja ei

tapahdu. Esimerkki pitkäkiertokyselystä listauksessa 6.2.

Listaus 6.2: Pitkäkiertokysely XMLHttpRequest-oliolla

```
var r = new XMLHttpRequest();
function longPoll() {
    r.open('GET', '/resurssi', true);
    r.onload = function () {
        console.log("Uusin tieto on: " + r.responseText);
        longPoll();
    };
    r.send();
}
```

Esimerkissä oletetaan, että palvelin vastaa vain, kun tiedot ovat muuttuneet. Uuden tiedon saatuaan selainsovelma avaa välittömästi taas uuden yhteyden. Olisi parempi, jos yhteyttä ei olisi tarpeen lopuksi sulkea, mutta tavanomaiseen ajoittaiseen kiertokyselyyn verrattuna pitkäkiertokysely silti harvemmin muuttuvan tiedon tapauksessa vähentää turhia HTTP-kutsuja. Usein muuttuvan tiedon tapauksessa HTTP-kutsujen määrä ei juuri muutu, mutta viive vähenee, sillä vastaus saadaan heti uuden tiedon tultua saataville. Myös pitkäkiertokysely kuitenkin tuhlaa tietoliikennekaistaa, sillä HTTP-otsakkeet lähetetään aina uudelleen uutta yhteyttä avattaessa. Varsinkin otsakkeisiin sisältyvät evästeet tuhlaavat kaistaa, sillä ne voivat olla useiden kilotavujen kokoisia.

XMLHttpRequest-olion käytön sijaan pitkäkiertokysely on mahdollista toteuttaa myös luomalla dynaamisesti `<script>`-tageja, joiden `src`-attribuutti viittaa palvelimeen. Palvelin voi vastauksena luoda javascript-komentosarjoja, jotka aiheuttavat uuden viestin välittävän funktiokutsun. Tällöin pitkäkiertokysely on mahdollista silloinkin, kun palvelimen verkkonimi poikkeaa sivun verkkonimestä. [Implementing Script Tag Long Polling for Comet Applications]

6.3 Ikuinen iframe

Iframe on HTML-elementti, jolla verkkosivuun voidaan sisällyttää toinen verkkosivu. Sisällytetty verkkosivu voi myös ajaa JavaScript-koodia ja kommunikoida funktiokutsujen kautta sisällyttävän verkkosivun kanssa. Tämän ansiosta iframeja voidaan käyttää myös jatkuvan palvelinyhteyden pohjana. Tekniikka perustuu siihen, että palvelin ei palauta sisällytetyn verkkosivun sisältöä heti, vaan jättää yhteyden auki. Pitkäkiertokyselyyn verrattuna vältetään siis uusien yhteyksien jatkuva avaaminen. Palvelin voi siten muuttuneiden tietojen tullessa saataville lisätä sisällytetylle sivulle JavaScript-pätkän, joka aiheuttaa funktiokutsun sisäl-

lyttävään sivuun. Toisien sanoen viitataan <iframe>-tagin <src>-attribuutilla dynaamiseen palvelinkomentosarjaan, joka lähettää <script>-tageja selaimelle, kun uutta tietoa tulee saataville. Palvelinkomentosarja ei sulje yhteyttä heti, vaan yhteys voi olla auki pidemmän aikaa. Palvelin lisää uusia <script>-tageja sitä mukaa, kun uusia tapahtumia halutaan lähettää selaimelle [Comet Programming].

Listauksessa 6.3 on esimerkki sisällyttävästä sivusta. Listauksessa 6.4 taas on esimerkki palvelimen vastauksesta ”/tiedot”-sivun sisällytetystä tietovirrasta.

Listaus 6.3: Sisällyttävä sivu

```
<html><body>
<script>
function uuttaTietoaSaatavilla(tiedot) {
    alert(tiedot);
}
</script>
<iframe src="/tiedot">
</body></html>
```

Listaus 6.4: Sisällytetty sivu

```
<html><body>
<script>
    parent.uuttaTietoaSaatavilla('Testi');
</script>

... Aikaa kuluu...

<script>
    parent.uuttaTietoaSaatavilla('Uutta tietoa saatavilla!');
</script>

... Yhteys edelleen avoinna odottamassa uutta tietoa ...
```

Ikuinen iframe on parannus pitkäkiertokyselyyn siinä, että tarvitaan vain yksi yhteys, joka jää taustalle odottamaan uusimpaa viestiä. Ratkaisuna se ei kuitenkaan ole kovinkaan elegantti, sillä se käyttää iframe-elementtiä sellaiseen tarkoitukseen, johon sitä ei alunperin oltu suunniteltu. Tästä aiheutuukin joissakin selaimissa ongelmia. Esimerkiksi iframen sisällöllä saattaa olla minimipituus, jota ennen uusia komentosarjatageja ei ajeta [Comet: Server-Push in Web]. Jotkin selaimet taas jättävät latauspalkin jatkuvasti näkyviin [The Future of Comet].

6.4 XMLHttpRequest multipart/x-mixed-replace

Tämä tekniikka toimii ainoastaan vanhemmissa Firefox-selaimissa. Uudemmissa versioissa tuki tekniikalle on poistettu, koska sen käyttö oli liian vähäistä [Bug 843508 - Remove support for multipart XHR responses]. Vanhentuneillakin tekniikoilla saattaa kuitenkin edelleen olla paikkansa, sillä kaikki käyttäjät eivät välttämättä aja uusimpia selainversioita.

Multipart/x-mixed-replace MIME-tyyppi ilmaisi, että saapuvassa tiedossa on monta osaa, joista jokainen korvaa edellisen. Jokaisella osalla saattoi olla oma MIME-tyyppinsä, vaikkapa image/jpeg. Tekniikkaa voidaan edelleen uudemminkin selaimissa käyttää kuvavirtojen kanssa, vaikka JavaScriptin XMLHttpRequestin kanssa sen käyttö ei enää onnistuisikaan. Jos esimerkiksi internet-yhteyteen kykenevä valvontakamera tarjoaisi uuden kuvan kerran sekunnissa osoitteessa `http://1.2.3.4/kuva`, voisi sen vastaus näyttää listauksen 6.5 kaltaiselta.

Listaus 6.5: Esimerkki multipart/x-mixed-replace-sisällöstä

```
HTTP/1.1 200 OK
Content-type: multipart/x-mixed-replace; boundary=erotin

--erotin
Content-type: image/jpeg

...kuva...
--erotin
Content-type: image/jpeg

...toinen kuva...
...
```

Kuvien välillä voi olla haluttu sekunnin viive. HTTP-kutsu valvontakameraan ei katkea, vaan selain jää odottamaan uusia osia näyttäen ne sitä mukaa kun ne saapuvat.

Osien MIME-tyyppi voi olla muutakin kuin kuvia. Listauksen 6.6 esimerkissä on Unix-komentotulkin komentosarjana toteutettu yksinkertainen multipart/x-mixed-replace -palvelin. Komentosarjan käynnistyttyä se jää odottamaan yhteyttä porttiin 1234. Avaamalla Firefox-selain osoitteeseen `http://localhost:2013` nähdään, että selain esittää palvelimen lähettämän tekstisisällön vaihdellen sanojen ”tick” ja ”tock” välillä sitä mukaa kun palvelimelta saadaan uutta sisältöä.

Listaus 6.6: Yksinkertainen multipart/x-mixed-replace-palvelin

```
(echo "HTTP/1.1 200 OK
Content-type: multipart/x-mixed-replace; boundary=erotin

--erotin"&&for i in {1..1000}; do echo "Content-type: text/plain

tick

--erotin
Content-type: text/plain

tock

--erotin"; done)|nc -l -i 1 1234
```

Ennen ominaisuuden poistamista FireFox-selaimesta, sitä voitiin käyttää luvussa 6.1 esitellyn XMLHttpRequest-olion kanssa, samaan tapaan kuin kierto-kyselyinkin tapauksessakin. Ikävä kyllä tuki tekniikalle vaihtelee huomattavasti selainten välillä. Vanhemmissa Firefox-selaimissa x-mixed-replace toimi, mutta Chrome- ja Safari-selaimissa sen käyttö ei ole koskaan ollut mahdollista [Comet: Server-Push in Web]. Sen käyttö onnistui listauksen 6.7 tapaan, kun tuki vielä oli FireFox-selaimessa.

Listaus 6.7: multipart/x-mixed-replace-sisällön vastaanottaminen

```
<script>
var r = new XMLHttpRequest();
r.multipart = true;
r.open('GET', '/', true);
r.onload = function () {
    console.log("Saatiin uusi dokumentti.");
};
r.send();
</script>
```

6.5 XMLHttpRequestin responseText-attribuutin valvonta

Pitkäaikaisen XMLHttpRequest-kutsun käyttö on mahdollista, vaikka multipart/x-mixed-replace ei olisikaan saatavilla. Tällöin menetetään automaattinen vastauksen pilkonta yksittäisiin viesteihin. Kuitenkin responseText-attribuutti edelleen päivittyy vähitellen tiedonsiirron edetessä, eikä vain yhdellä kertaa kutsun lopussa. Tämän ansiosta XMLHttpRequest-oliota voidaan käyttää jatkuvaan yhtey-

teen multipart/x-mixed-replace-tuen puuttumisesta huolimatta. Tekniikka tunnetaan myös lyhemmällä nimellä ”XHR streaming” [The Future of Comet].

Selain kutsuu XMLHttpRequest-olion onreadystatechange-metodia silloin, kun palvelimelta saadaan uutta tietoa. Vanhemmissa Internet Explorer -selaimen versioissa responseText-attribuuttia ei voida lukea ennen koko vastauksen saapumista [COMET Streaming in Internet Explorer]. Tekniikan toisena heikkoutena XMLHttpRequest-olion responseText-attribuutti kasvaa rajatta. Koska attribuutti on vain luettavissa, ainut tapa nollata se on aloittaa ajoittain uusi yhteys. Siirrettävän tiedon alussa täytyy olla tietty määrä täytettä, sillä jotkin selaimet eivät tuota tapahtumia ellei tavuja ole saapunut minimimäärä. Esimerkiksi jotkin Safari-selaimen versiot eivät käynnistä onreadystatechange-kutsuja, ellei tietoa ole saapunut vähintään 256 tavua [The Future of Comet].

6.5.1 XMLHttpRequest ja ”chunked”-vastaukset

Ongelmana XMLHttpRequestin responseText-attribuutin valvontaa varten tuotettavassa sisällössä on, että palvelimen olisi periaatteessa annettava sisällön koko Content-Length -otsakkeessa [RFC2616 4.4]. Sisällön kokoa ei kuitenkaan yleensä voida tietää XMLHttpRequestin responseText-attribuutin valvontaa varten käynnistetyssä pitkäkestoisessa yhteydessä ennalta. Chunked on HTTP:n versiossa 1.1 esitelty vastaustyyppi, jossa palvelin voi lähettää vastauksen paloittain ilman Content-Length -otsaketta [RFC2616 3.6.1]. Palvelin ilmaisee haluavansa lähettää chunked-tyyppisen vastauksen asettamalla ”Transfer-Encoding”-otsakkeen arvoksi ”chunked”. Listauksessa 6.8 on Unix-komentoriviesimerkki, joka ajon jälkeen jää odottamaan yhteyttä porttiin 1234.

Listaus 6.8: Chunked-koodatun vastauksen tuottava palvelinesimerkki

```
(echo "HTTP/1.1 200 OK
Content-Type: text/plain
Transfer-Encoding: chunked

3
hei
b
ppa maailma
0

")|nc -l -i 1 1234
```

Avaamalla selain osoitteeseen `http://localhost:1234` nähdään, että palois- ta koostettiin vastaus ”heippa maailma”. `Content-Length` -otsakkeen lähettäminen ei siis ole välttämätöntä. Palojen välissä voisi kulua pidempikin aika, joten chunked-koodaus sopii hyvin `XMLHttpRequest`in kanssa käytettäväksi. Yhteys voitaisiin jättää auki, jolloin ajoittain `responseText`-attribuuttia tutkimalla saataisiin tietoon uudet palvelimelta tulleet viestit. Vastauksen loppuminen ilmaistaan nollakokoisena palana. Pelkkä TCP-yhteyden katkaiseminen ei olisi riittävä tapa kertoa vastauksen loppumisesta, sillä silloin ei tiedettäisi katkesiko yhteys virheen vuoksi kesken lähetyksen, vai oliko yhteyden katkaiseminen tarkoituksellista.

6.6 EventSource API

`EventSource` on rajapinta, jonka avulla JavaScript-ohjelma voi lähettää viestejä ja tapahtumia palvelimelta virtana. Koska `EventSource` on erityisesti suunniteltu comet-tarkoitukseen, sen käyttö on `XMLHttpRequest`-oliota luontevampaa `EventSource`-rajapinnan ollessa saatavilla.

Kun `EventSource`-olio luodaan, ottaa se yhteyden palvelimeen, joka välittää viestit selaimen MIME-tyypillä ”text/event-stream”. Yksinkertaisimmillaan formaatti koostuu tekstiriveistä, jotka alkavat merkkijonolla ”data:”, jonka jälkeen tulee itse viesti. Listauksessa 6.9 on yksinkertainen ”text/event-stream” MIME-tyyppiä tarjoileva palvelin Unix-komentotulkin komentosarjana. Todellisessa käyttötilanteessa palvelin pitäisi yhteyden auki pysyvästi, jotta sen kautta voidaan lähettää uusia viestejä niiden tullessa saataville.

Listaus 6.9: text/event-stream -palvelin

```
(echo "HTTP/1.1 200 OK
Content-Type: text/event-stream

data: Esimerkkiviesti.

data: Toinen esimerkkiviesti.

data: Viesti voi jatkua
data: useammallekin riville.

")|nc -l -i 1 1234
```

Esimerkkipalvelimen kanssa kommunikoiva selainkomentosarja on myöskin hyvin yksinkertainen, kuten listauksesta 6.10 nähdään.

Listaus 6.10: EventSource-esimerkkiohjelma

```

<html><body>
<script>
var eventSource = new EventSource('source');
eventSource.onmessage = function (e) {
    document.write('Viesti: ' + e.data);
}
</script>
</body></html>

```

Viestin lisäksi palvelin voi lähettää myös tapahtumia, joilla on data-osan lisäksi myös tapahtuman nimi. Nimet ilmaistaan "text/event-stream"-formaatissa ennen data-rivejä tulevalla event-rivillä. Tällöin selainkomentosarjassa palvelimelta saapuvat tapahtumat käsitellään samaan tapaan, kuten käyttöliittymän DOM-tapahtumatkin. Funktiolla `addEventListener` assosioidaan haluttu tapahtuma sen käsitelijäfunktioon [Server-Sent Events]. Esimerkki tapahtumia tuottavasta palvelimesta listauksessa 6.11 ja käsittelevästä selainkomentosarjasta listauksessa 6.12.

Listaus 6.11: Tapahtumia tarjoileva text/event-stream -palvelin

```

(echo "HTTP/1.1 200 OK
Content-Type: text/event-stream

event: eka
data: Esimerkkidataa.

event: toka
data: Toinen esimerkkidata.

event: eka
data: Bonusesimerkki samalle tapahtumalle.

")|nc -l -i 1 1234

```

Listaus 6.12: Esimerkki EventSource-tapahtumista

```

<html><body>
<script>
var eventSource = new EventSource('source');
eventSource.addEventListener('eka', function (e) {
    alert('Eka: ' + e.data);
});
eventSource.addEventListener('toka', function (e) {
    alert('Toka: ' + e.data);
});
</script>
</body></html>

```

EventSourcen ollessa saatavilla se on luontevampi vaihtoehto viestintään kuin XMLHttpRequest, sillä se on suunniteltu varta vasten pitkäaikaiseen yhteyteen comet-viestien välittämiseksi. Rajapinta on suosituimpien selainten uusissa versioissa käytettävissä, lukuun ottamatta Internet Explorer -selainta [Server-Sent Events].

6.7 WebSocket

WebSocket on protokolla, joka mahdollistaa jatkuvan kaksisuuntaisen yhteyden palvelinohjelman ja asiakasohjelman välillä. Tekniikasta näyttää vähitellen muodostuvan standardiratkaisu comet-tarkoituksiin, joten perehdytään siihen muita tarkemmin. EventSourcseen verrattuna WebSockettien etuna myös selainsovelmalla voi lähettää uutta yhteyttä avaamatta palvelimelle viestejä, kun EventSourcessa viestintä onnistui vain palvelimelta selaimelle. Kumpikin puoli voi siis koska tahansa lähettää tietoa vastapuolelle kumpaankin suuntaan saman sovikeyhteyden yli. Yhteyttä ottava asiakasohjelma voi ilmaista haluavansa aloittaa WebSocket-protokollan käytön HTTP-otsakkeiden kautta. Saman WWW-palvelimen samassa portissa voidaan siten verkkosivujen lisäksi palvella myös WebSocket-kutsut [RFC6455 1.3.].

Protokollaa voi toki käyttää muukin asiakasohjelma kuin WWW-selain, mutta keskitytään tässä selaimen tapaukseen. WebSockets vähentävät huomattavasti viivettä ja verkkoliikennettä verrattuna aikaisempiin JavaScriptin vaihtoehtoihin, kuten kiertokyselyyn ja pitkäkiertokyselyyn [websocket.org]. Saman yhteyden käyttö kumpaankin suuntaan voi tuoda huomattavan säästön, sillä uusien TCP-yhteyksien avaaminen on aikaavievää kolmitiekättelyn vuoksi [RFC793, 12]. Vaikka usein käytännössä selain osaisi muutenkin välttää uuden TCP-yhteyden avaa-

misen HTTP/1.1-standardin esittelemän ”Connection: Keep-Alive” -otsakkeen ansiosta, säästetään silti ainakin otsakkeiden (erityisesti evästeiden) lähettämiseen kuluva tietoliikennekaista. Koska WebSocketit rasittavat palvelimia vähemmän, riittää yksi palvelin entistä useamman kutsun palvelemiseen.

Selainten WebSocket-tuki on vähitellen paranemassa, mutta useat suositut selainversiot eivät vielä tue sitä. Ennen selainten tuen paranemista ei pelkkien WebSockettien varaan verkkopalvelua voi siis vielä rakentaa. Tarvittaessa on turvaututtava vanhempiin ratkaisuihin, joten pitkäaikaisyyhteyden toteuttaminen monimutkaistuu. Olisi suoraviivaisempaa, jos selainsovelmasta voitaisiin automaattisesti käyttää parasta mahdollista yhteystapaa. Ohjelmoijan avuksi onkin saatavilla kirjastoja, jotka abstrahoivat yhteyden muodostamisen siten, että yhteystavasta huolimatta voi selainsovelman ohjelmakoodi pysyä samana. Tällaisia kirjastoja ovat esimerkiksi sock.js [sock.js] ja socket.io [socket.io]. Kirjastoja käytettäessä saattaa yhteys kulissien takana perustua WebSockettiin, mutta se voi tarvittaessa olla muullakin tavoin toteutettu, jos tukea ei ole saatavilla. Esimerkiksi Socket.io-kirjasto tukee WebSockettien lisäksi sovikeyhteyksiä Adobe Flashin kautta, tavalista kiertokyselyä, pitkäkiertokyselyä, multipart-virtaa ja iframe-tagien käyttöä [socket.io].

6.7.1 Yhteyden muodostuminen

Kun yhteys avataan, on tietynlaisen kättelyn tapahduttava, jotta siirtyminen WebSocket-yhteyteen onnistuu. Selain pyytää WebSocket-protokollan käyttöä lähettämällä ”Upgrade: websocket”-otsakkeen [RFC6455 1.3.]. Kättelyä on lisäksi tarkoituksella monimutkaistettu HTTP-otsakkeiden sisältöön perustuvalla kryptografisen SHA-1 -hajautusarvion laskennalla, jotta yhteys onnistuisi vain kun kummallakin osapuolella todella on WebSocket-valmius. Protokollan kättely on siis suunniteltu niin, että sillä ei ole mahdollista ottaa yhteyttä esimerkiksi SMTP-sähköpostipalvelimeen, joka ei tällaista yhteyttä odota [RFC6455 1.6.]. Ilman tätä estoa voisi WWW-sivulle sisällytetty JavaScript-haittaohjelma ottaa WebSockettien avulla TCP-yhteyksiä käyttäjän tietämättä. Koska käyttäjällä voi olla yhteys paikallisverkkonsa laitteisiin, joihin Internetistä ei palomuurin vuoksi muuten olisi pääsyä, olisi tällaisten yhteyksien salliminen vaarallista.

WebSocket pyrkii olemaan mahdollisimman lähellä puhdasta sovikeyhteyttä, tehden vain sellaiset lisäykset, joilla yhteys saadaan toimimaan nykyisten selaintekniikoiden kanssa [RFC6455 1.5.]. Se toimii tasona puhtaiden TCP-sovikkeiden yllä, varmistaen väärinkäytön estämiseksi, että eri palvelinten (”origin”) väliset

kutsun onnistuvat vain, jos vastaanottaja ne hyväksyy. Selain sallii JavaScriptistä avatun yhteyden vain, jos URLissa on sama skeema, omistaja ja portti [HTML5Nightly 5.3].

Yhteyden avaaminen alkaa siis samalla tavalla kuten HTTP/1.1-versiossakin. Selain lähettää palvelimelle halutun resurssin polun. Yhteyttä avatessa myös ”host:”-otsake edelleen välittyy, joten sama palvelin voi myös WebSocketia käytettäessä palvella samassa IP-osoitteessa useita eri omistajanimiä [RFC2068 14.23]. Esimerkiksi ”example.com” ja ”uta.fi” voisivat siten olla samalla palvelimella, joilla kummallakin saattaisi olla esimerkiksi ”/chat”-resurssi.

6.7.2 Tiedonsiirto

Kun kättely on suoritettu, vaihdetaan HTTP-protokollasta WebSocket-protokollaan. Vaihdon jälkeen voi kumpikin osapuoli lähettää WebSocket-kehymiä vapaasti toisilleen. WebSocketilla on kaksi kehystystapaa sisällöstä riippuen: binääri- ja tekstikehystys. Binäärikehykset kertovat alussa siirrettävän tietokehymksen koon, kun taas tekstikehykset alkavat nollatavulla ja loppuvat 0xFF-tavuun [websocket.org]. Yksittäinen WebSocket-viesti voi koostua useista kehyksistä. Kehyksien ensimmäinen bitti kertoo, tuleeko samaan viestiin liittyviä kehymiä vielä lisää vai onko kyseessä viimeinen kehys [Beams 2012, 40:30-].

Kun sovellus haluaa lähettää tietoa toiseen sovikkeeseen, abstrahoi TCP-protokolla sovellukselle tietojen pilkkomisen IP-paketteihin ilman, että sovelluksen täytyisi ottaa kantaa verkkoliikenteen yksityiskohtiin. Sovelluksen näkökulmasta sovikkeeseen voidaan siis vapaasti kirjoittaa ja lukea, ilman että sen täytyisi ottaa huomioon verkkoliikenteen yksityiskohtia, kuten mahdollisten matkalla kadonneiden pakettien uudelleenlähetystä, tai väärässä järjestyksessä saapuneiden pakettien järjestelyä. WebSocket-protokolla lisää näiden TCP-segmenttien päälle vielä oman kehyksensä.

Kun HTTP-otsakkeisiin perustuvan kättelyn jälkeen vaihdetaan WebSocket-protokollaan, herää kysymys miksi tällainen kehystys on tarpeen. Periaatteen mukaan soviketta voitaisiin suoraviivaisemmin ja tehokkaammin käyttää pelkkänä TCP-yhteytenä protokollan vaihtamisen jälkeen.

Eräs syy kehukseen on sen mahdollistama siirrettävän tiedon mutaatio, jonka avulla estetään välityspalvelimen välimuistin myrkytykseen perustuva hyökkäys (”cache poisoning”). Hyökkäys perustuu välityspalvelimiin, jotka tutkivat HTTP-liikenteeksi luulemaansa WebSocket-liikennettä ja yrittävät tallentaa vastauksia virheellisesti ”Host:”-otsakkeen perusteella välimuistiinsa. Tämä on vaarallista,

koska samaa välimuistia saattavat käyttää esimerkiksi kaikki saman yrityksen työntekijät. Jos hyökkääjä voisi tallentaa haluamiaan tietoja vapaasti välimuistiin, olisi koko yrityksen tietoturva uhattuna. [Huang]

Välimuistin myrkytyksen estämiseksi WebSocket-kehyksessä voi olla satunnainen vaihtuva kenttä, jonka perusteella siirrettävät tavut muutetaan toisiksi siten, että vastaanottaja voi edelleen kehyksen tietojen perusteella palauttaa XOR-operaatiolla tavut alkuperäisiksi [RFC6455, 5.3]. Koska tiedot jatkuvasti muuttuvat, ei hyökkääjä voi itse valita siirtyviä tavuja. Välityspalvelimen kannalta HTTP-kutsuilta vaikuttavien WebSocket-kehysten luonti ei siten enää hyökkäyksenä onnistu.

Toinen syy lisäkehukseen on datavirran jako viesteihin. Vaikkakin sovelluksen kannalta katkoton datavirtakin olisi riittävä, on useiden käytännön sovellusten kuitenkin pilkottava virta yksittäisiin viesteihin. Esimerkiksi pikaviestikeskusteluohjelma voisi toki toimia katkottoman datavirrankin perusteella, mutta käytännössä se joutuisi silloin itse tekemään jonkinlaisen jaon datavirrasta pienempiin viesteihin. On kätevää, jos pilkonta tapahtuu jo valmiiksi. Kun uusi WebSocket-kehys saapuu, onmessage-funktio saa kutsun, joten ohjelmoijan ei tarvitse huolehtia datavirran paloittelusta [WebSocket API, 5].

Kolmantena syynä kehystykseen WebSocket-protokolla lisää kehüksensä avulla oman sulkemiskättelyn tavanomaisen TCP:n FIN/ACK-paketin lisäksi. Lisäys tehtiin, koska TCP:n sulkemiskättely ei aina ole luotettava varsinkaan välityspalvelimien yli [RFC6455 1.4.].

6.7.3 Palomuurit ja välityspalvelimet

Jotta yhteys onnistuisi, on IP-pakettien välityttävä ongelmitta selaimelta palvelimelle ja takaisin. Välissä on laitteita, jotka saattavat tutkia paketteja, ja hylätä liikenteen jos tietyt ehdot täyttyvät. Tällaisia laitteita ovat palomuurit ja välityspalvelimet.

Palomuurit hylkäävät liikenteen, joka ei saavu sallittuun porttiinumeroon. Koska WebSocket-yhteys alkaa tavallisena HTTP- tai HTTPS-porttisena yhteytenä, jossa vain protokolla vaihtuu yhteyden aikana, ei palomuurilla ole syytä hylätä liikennettä portin perusteella, sillä kättely ja yhteys kättelyn jälkeen tapahtuvat kumpikin HTTP- tai HTTPS-porttiin.

Välityspalvelimet eivät välttämättä salli WebSocket-kättelyn onnistumista, tai eivät välitä kättelyn jälkeisiä WebSocket-kehysiksiä, sillä yhteydet ovat auki pitkään, eivätkä näytä HTTP-liikenteeltä. Yhteyden onnistumista voidaan parantaa

siirtämällä WebSocket-liikenne salattuna. Kun välityspalvelimet eivät näe siirtyviä tavuja, ne eivät myöskään osaa hylätä välittämäänsä salattua tietovirtaa, joka ei purettuna näyttäisi HTTP-liikenteeltä. (Beams 2012, 19:30-.)

6.7.4 WebSocket-esimerkkiohjelma

Tornado on Python-verkkokirjasto, joka tukee myös WebSoketteja. Tornadon avulla yksinkertaisen WebSocket-kutsuja palvelevan WWW-palvelimen pystyttäminen onnistuu varsin pienellä koodimäärällä. Listauksessa 6.13 on esimerkki Tornadolla toteutetusta palvelinohjelmasta, joka ottaa vastaan WebSocket-yhteyksiä. Kun yhteyden kautta saadaan viesti, tulostetaan sen sisältö. Viestin käsittelyn loppuksi lähetetään kutsujalle vastauksena ”Kiitos”.

Listaus 6.13: WebSocket-palvelin Tornadolla

```
import tornado.ioloop
import tornado.web
import tornado.websocket

class MainHandler(tornado.websocket.WebSocketHandler):
    def on_message(self, message):
        print "Sain viestin:", message
        self.write_message("Kiitos")

application = tornado.web.Application([
    (r"/", MainHandler),
])

application.listen(8888)
tornado.ioloop.IOLoop.instance().start()
```

Edellisen esimerkkiohjelman kanssa kommunikoivalta JavaScript-asiakasohjelmalta vaaditaan WebSocket-yhteyden avaamista, uusiin viesteihin reagoimista ja sovikkeeseen kirjoittamista. Listauksessa 6.14 esimerkkiohjelma, joka esittelee kyseiset toiminnot.

Listaus 6.14: Yhteyden muodostaminen Tornado-palvelimeen

```

<html><body>
<script>
var socket = new WebSocket('ws://localhost:8888')
socket.onmessage = function (messageEvent) {
    alert('Sain viestin: ' + messageEvent.data);
}
socket.send('Moro');
</script>
</body></html>

```

6.8 WebRTC

Viimeisenä comet-tekniikkana esitellään WebRTC (Web Real Time Communication), jota käytetään yleisimmin P2P-videokonferenssisovelluksien toteuttamiseen. Käyttö myös comet-viestintään voisi kuitenkin olla mahdollista, mikäli palvelin voisi toimia yhtenä P2P-kommunikoinnin vastapuolena. Kirjoitushetkellä tällaiseen käyttöön soveltuvia palvelintoteutuksia ei vielä ollut tarjolla. Kunhan palvelinpuolen tuki kehittyy, WebSockettien mahdollistamien TCP-yhteyksien lisäksi voitaisiin palvelimen kanssa viestiä JavaScriptillä myös UDP:n yli. UDP-yhteyksissä siirtokanavan luotettavuuden sijaan keskitytään latenssin minimointiin, joka etenkin nopeatempoisia verkkopelejä toteuttaessa on järkevä valinta.

WebRTC on kokoelma teknologioita, joiden avulla JavaScriptillä toteutetut selainsovelmat voivat kommunikoida reaaliajassa toistensa kanssa verkon yli. WebRTC tarjoaa tavan pakata käyttäjän verkkokameran kuvaa, ja lähettää se reaaliaikaisena virtana toiselle käyttäjälle. Yhteyttä voidaan käyttää muuhunkin, kuten äänipuheluihin. Sitä voidaan käyttää myös vapaadatan, kuten tietokonepelien liiketietojen, siirtoon [WebRTC Overview]. WebRTC vahvistaa JavaScriptin asemaa, sillä ennen nämä olivat mahdollisia toteuttaa vain selaimen lisäosilla, kuten Flashilla.

Toisin kuin selainsovelmissa yleensä, WebRTC:n kautta voidaan kommunikoida suoraan toiselle käyttäjälle. Tiedon ei siis tarvitse mennä palvelimen kautta. Tämä on mahdollista, vaikka käyttäjät olisivat IP-osoitteita muuttavan NATin takana. NAT ("Network Address Translation") on käytössä tilanteissa, joissa esimerkiksi kotiverkossa on useita laitteita, mutta ulospäin kaikilla laitteilla on sama IP-osoite. WebRTC koordinoi kommunikoinnin NATin takana sijaitsevien laitteiden kanssa joko suoraan tai tarvittaessa välityspalvelinta käyttäen. [Getting Star-

ted with WebRTC]

WebRTC koostuu kolmesta pääasiallisesta APIsta: `MediaStream`, `RTCPeerConnection` ja `RTCDataChannel` [Google I/O 2013].

6.8.1 `MediaStream`

Video- ja/tai äänivirtaa edustavan `MediaStream`-olion saa `navigator.getUserMedia()`-kutsulla. Selaimesta riippuen `getUserMedia`-funktiolla voi olla selainkohtainen etuliite. Listauksessa 6.15 on esimerkki virran saamisesta ottaen etuliitteet huomioon. Esimerkki toistaa käyttäjän webbikameran virran `<video>`-elementissä, mikäli käyttäjällä on kamera ja hän hyväksyy sen käyttöönoton.

Metodille `getUserMedia` annetaan ensimmäisenä argumenttina rajoite, joka kertoo millaista mediavirtaa halutaan. Rajoite voi kertoa (kuten esimerkissä), että halutaan videovirta, mutta se voi myös tarkemmin rajata myös halutun virran resoluution. Kaksi viimeistä argumenttia ovat funktiot, joita kutsutaan kun virta saadaan onnistuneesti, tai jos mediavirtaa ei saatu virheen vuoksi.

Listaus 6.15: `getUserMedia`-funktion käyttö

```
<html><body>
<script>
document.write('<video autoplay></video>');
navigator.getMedia = navigator.getUserMedia || navigator.
    webkitGetUserMedia
    || navigator.mozGetUserMedia || navigator.msGetUserMedia;

navigator.getMedia(
    {video: true},
    function (stream) {
        var video = document.querySelector('video');
        video.src = window.URL.createObjectURL(stream);
    },
    function (error) {
        console.log(error);
    }
);
</script>
</body></html>
```

Ääntä puolestaan voi käsitellä `getMedia`-funktion rajoiteargumentilla `{audio: true}`. Webbikameran sisällön sijaan on myös mahdollista pyytää virta käyttäjän ruudun sisällöstä, tosin toiminto oli vielä kokeellinen kirjoitushetkellä uusimmas-

sa Chrome-selaimen versiossa 30, vaati erityisasetuksia ennen käyttöä. Toiminto mahdollistaa ruudunjaon netin yli, esimerkiksi kun halutaan esitellä jonkin ohjelmiston toimintaa videokonferenssissa. Ruutukaappaus on mahdollista, kun `getUserMedia`-funktiolle annetaan rajoitteeksi `{chromeMediaSource: 'screen'}`.

6.8.2 RTCPeerConnection

`RTCPeerConnection` mahdollistaa suoran yhteyden toiseen käyttäjään. Yhteyden yli voi lähettää audiota, videota tai muuta dataa [Google I/O 2013]. `RTCPeerConnection` huolehtii videon ja audion pakkauksesta koodekillalla, suhteuttaen siirrettävän datan määrän käyttäjälle saatavilla olevaan kaistanleveyteen. `RTCPeerConnection`illa on muitakin tehtäviä, kuten yhteyden muodostaminen vaikka käyttäjä olisi NATin takana [WebRTC 1.0]. NATin yli kommunikointia varten kehitettyä tekniikkaa kutsutaan nimellä ICE ("Interactive Connectivity Establishment").

Laitteen sijaitessa NATin takana poikkeaa sen IP-osoite julkisesta IP-osoitteesta. Jos paikallinen IP-osoite annettaisiin Internetin yli vastapuolelle, ei siitä olisi vastapuolelle hyötyä, koska kyseinen IP-osoite toimisi vain paikallisverkossa. Vastapuolelle on siis voitava antaa laitteen julkinen IP-osoite. Eräs tapa löytää osoite olisi ottaa yhteyttä Internetissä sijaitsevaan palveluun, joka voisi vastauksena kertoa kutsujan julkisen IP-osoitteen. STUN ("Session Traversal Utilities for NAT") on protokolla, jonka avulla voidaan tarjota juurikin tällainen palvelu. Siinä STUN-asiakasohjelma ottaa yhteyden STUN-palvelimeen, joka antaa vastauksena asiakkaan julkisen IP-osoitteen.

Useimmissa tapauksissa kommunikointi suoraan toiselle käyttäjälle onnistuu ICE:n avulla, vaikka kumpikin osapuoli olisi NATin takana. Jos suora yhteys ei ole mahdollinen, ICE turvautuu TURN-välityspalvelimen ("Traversal Using Relay NAT") käyttöön. ICE toimii kuitenkin parhaiten, kun NATit käyttäytyvät siten, että suora yhteys onnistuu. Tällöin audiosiiirron tapauksessa äänenlaatu paranee, yhteyden rakentaminen on nopeampaa ja yhteys käyttää vähemmän verkkoliikennespesseja [RFC5245 20.1.]. ICE kokeilee systemaattisesti läpi erilaisia yhteysvaihtoehtoja, kunnes se löytää toimivan yhteyskeinon [RFC5245, 9]. Vaihtoehtoina ovat vastapuolen suora paikallisverkko-osoite, NATin julkinen osoite tai TURN-palvelimen osoite.

6.8.3 RTCDataChannel

RTCDataChannel-oliot edustavat WebRTC-tietokanavaa, jonka yli voidaan siirtää vapaadataa. Kanava luodaan RTCPeerConnectionin metodilla createDataChannel, jolle annetaan argumenttina halutun kanavatyypin kuvaus. Kuvauksen perusteella voidaan valita joko UDP- tai TCP-pohjainen kanava. UDP saadaan käyttöön pyytämällä kuvauksessa epäluotettava kanava. Kun kanava on luotu, voidaan sen yli lähettää tietoa send-metodilla.

7 YHTEYDET JAVA-SOVELMISTA

Java kehitettiin alunperin sulautettujen järjestelmien tarpeisiin, mutta mukautettiin soveltumaan myös WWW-käyttöön. Sanalla ”Java” voidaan tarkoittaa sekä Java-virtuaalikonetta (Java Virtual Machine, JVM) että Java-kieltä. Java-virtuaalikone suorittaa Java-kielestä käännettyä Java-tavukoodia, mutta muitakin Java-tavukoodille käännettäviä kieliä on olemassa. Wikipedia listaa [Wikipedia:JVM languages] merkittävimpinä esimerkkeinä kielet Clojure, Groovy, Scala, JRuby, Jython ja Rhino. Näistä erityisesti vuonna 2003 julkaistu funktionaalinen kieli Scala näyttää saaneen suuren suosion. Siinä on Java-kieleen verrattuna useita hyödyllisiä parannuksia ja lisäominaisuuksia, kuten anonyymit funktiot (lambdat) ja listakomprehensiot.

Kun Javaa käytetään WWW-ohjelmointiin, voidaan sitä ajaa joko palvelimella tai käyttäjän selaimessa lisäosana. Palvelimella ajettaessa Java-sovelma toimii yhteyttä ottavan selaimen näkökulmasta kuten mikä tahansa WWW-sovellus, vastaten HTTP-kyselyihin samaan tapaan, kuten muillakin kielillä toteutetut sovellukset. Java-palvelinsovelmia kutsutaan myös servleteiksi. Esimerkiksi Tampereen yliopiston Informaatiotutkimuksen laitoksen (nykyisin Informaatiotutkimuksen ja Interaktiivisen Median TRIM-tutkimuskeskus) tiedonhaun opetus- ja tutkimusohjelma QPA (Query Performance Analyser) laadittiin Java-servlettinä [Sormunen and Pennanen, 2004, figure 1].

Selaimessa ajettuna Java-sovelma voi viestiä palvelimen kanssa TCP-protokollalla suoraan Javan vastakkeiden (socket) avulla, tai niiden päälle rakennettujen muiden palvelujen, kuten JDBC:n tai Java RMI:n kautta. JDBC (”Java DataBase Connectivity”) on tapa olla yhteydessä tietokantaan verkon yli. Java RMI (”Java Remote Method Invocation”) taas on tapa kutsua verkon yli metodeja, pitäen sisällään tietojen muuttamisen verkossa siirrettävään muotoon (tiedon sarjoitus) ja roskienkeruun [Wikipedia:RMI]. TCP:n lisäksi myös UDP-pakettien käyttö Java-sovelmasta on mahdollista. Muiden IP-protokollien kuten ICMP:n käyttö ei kuitenkaan Java-sovelmasta onnistu [Harold, 2004, section 13.1.]. Eräs lisärajoitus selaimessa ajettaville Java-sovelmille estää niitä ottamasta yhteyksiä muille palvelimille kuin sille, jolta sovelma alunperin haettiin. Rajoitusta voidaan kiertää luomalla välityspalvelin, joka välittää Java-sovelman pyynnön eteenpäin [Chase, 2004].

7.1 Javan vastakkeet (sockets)

Yhteydenotto selaimen Java-sovelmasta palvelimeen tapahtuu luomalla uusi vastake Socket-luokan muodostinta (englanniksi ”constructor”) käyttäen. Kaikki Javan tarjoamat TCP-yhteyksiä käyttävät palvelut hyödyntävät tosiasiaassa alimmalla tasolla Socket-luokkaa. Yhteyden muodostuttua voidaan luodulta Socket-luokan oliolta pyytää syöte- ja tulostevirtoja (stream) edustavat oliot, joiden kautta varsinainen viestintä sitten tapahtuu.

Listauksessa 7.1 on lyhyt esimerkki, jossa avataan TCP-yhteys Tampereen yliopiston WWW-palvelimelle. Yhteyden avauksen jälkeen sen yli luetaan pääsivun sisältö HTTP-protokollaa käyttäen.

Listaus 7.1: Java-sovikkeen avaaminen

```
Socket uta = new Socket("www.uta.fi", 80);
OutputStream raw = uta.getOutputStream();
OutputStream buffered = new BufferedOutputStream(raw);
out = new OutputStreamWriter(buffered, "ASCII");
out.write("GET / HTTP/1.0\r\n\r\n");
out.flush();
BufferedReader in = new BufferedReader(
    new InputStreamReader(uta.getInputStream())
);
String line = null;
while ((line = in.readLine()) != null) {
    System.out.println(line);
}
```

Esimerkissä avataan palvelimen HTTP-portti, lähetetään sinne pääsivun pyyntö, luetaan tulostusvirrasta palvelimen antama vastaus ja tulostetaan se rivi kerrallaan. Socket-luokkaa vastaava luokka UDP-pakettien lähettämistä varten on DatagramSocket.

7.2 JDBC

Monen WWW-sovelluksen päätehtävänä on luoda, lukea, päivittää tai tuhota palvelimen tietokannan taulujen rivejä. JavaScript-sovelmissa tämä toteutetaan yleensä laatimalla tietokannan päivitykseen sopiva HTTP-rajapinta. Samaa tapaa voidaan toki käyttää Java-sovelmistakin, mutta JDBC-ohjelmointirajapinnan (API) ansiosta myös suora yhteys tietokantaan on mahdollinen. Se mahdollistaa kyselyt relaatiotietokantoihin verkon ylitsekin.

Listauksessa 7.2 otetaan yhteys localhost-paikallisosoitteessa sijaitsevaan MySQL-tietokantaan. Yhteyden yli luetaan tietokannasta yksittäinen numero. Yhteyden ottamiseksi tarvitaan tietokannan URL, josta käy ilmi tietokannan tyyppi, tietokantapalvelimen verkkonimi ja halutun tietokannan nimi. URLin lisäksi tarvitaan tietokantapalvelimen käyttäjän nimi ja salasana.

Listaus 7.2: Yhteyden avaaminen MySQL-tietokantaan

```
import java.sql.*;

public class Example {
    public static void main(String[] args) {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conn = DriverManager.getConnection(
                "jdbc:mysql://localhost/database_name",
                "user",
                "password"
            );
            Statement statement = conn.createStatement();
            String sql = "SELECT id FROM table";
            ResultSet resultSet = statement.executeQuery(sql);
            resultSet.next();
            System.out.print("ID: " + resultSet.getInt("id"));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Todellisessa käyttötilanteessa olisi poikkeustilanteiden käsittely ja tietokantayhteyden sulkeminen myös huomioitava [tutorialpoint]. Esimerkin toimimiseksi on paikalliskoneella oltava MySQL, sekä sen JDBC-liitin asennettuna.

7.3 RMI

RMI (Remote Method Invocation) on ohjelmointirajapinta ja Javan luokkakirjasto, jolla voidaan käyttää palvelimella sijaitsevia olioita. Metodikutsut toimivat ohjelmoijan kannalta samaan tapaan, kuin jos ne olisivat osa paikallista sovellusta. Tietenkään palvelimella sijaitsevien metodien kutsuminen ei voi olla yhtä nopeaa kuin paikallisten metodien kutsuminen. Kutsuttaessa on aina myös vaurauduttava poikkeuksiin (exception), joita voi syntyä mikäli kutsu epäonnistuu verkkoyhteysongelmien vuoksi [Harold, 2004, section 18.1.]. Koska myös RMI on

rakennettu vastakkeiden päälle, koskee sitäkin rajoitus, joka estää ottamasta yhteyttä muihin kuin siihen palvelimeen, jolta itse Java-sovelma on ladattu.

7.3.1 RMI-palvelun osat

Minimaalinen RMI-ohjelma vaatii neljä osaa:

- olio
- rajapinta
- palvelinohjelma
- asiakasohjelma

Olio sisältää toiminnot, jotka halutaan ajaa palvelimella. Jotta luokkaa voitaisiin käyttää palvelemaan etäkutsuja, täytyy se periyttää `UnicastRemoteObject`-luokasta. Jotta asiakasohjelma tietäisi millaisia metodeja oliossa on, tarvitaan itse olion lisäksi sen rajapinnan määrittelmä [Schildt 2011, RMI].

Jotta palvelimella sijaitsevaa oliota voitaisiin kutsua, on siitä jollakin tapaa luotava instanssi, johon palvelin voi sitten tarjota asiakkaalle viitteen. Instanssin luomista varten ajetaan palvelinohjelma, joka saa viitteen olioon ja tallettaa sen palvelimella ajettavaan ”rmiregistry”-daemoniin. Rmiregistry välittää viitteen tarvittaessa asiakasohjelmille, jotka voivat sitten viitteen kautta käyttää palvelimella ajettavia toimintoja.

7.3.2 Rmiregistry

Kun asiakasohjelma haluaa kutsua etämetodeja, on ensin saatava viittaus ensimmäiseen olioon palvelimella. Rmiregistry on palvelinohjelma, joka toimii oletuksena portissa 1099 ja kertoo tarkemmat tiedot halutun olioviitteen saamista varten. Java-ohjelma saa olioviitteen, kunhan se tietää halutun palvelun Rmiregistryn URLin. Jos esimerkiksi kyseessä olisi aurinkokunnan planeettatietoja tarjoava olio, voisi asiakasohjelma saada sen rajapintaan viitteen listauksessa 7.3 esitetyllä tavalla [rmiregistry]. Kun viite on saatu, voidaan metodeja kutsua kuin kyseessä olisi paikallisesti toteutettu olio.

Listaus 7.3: Rajapintaviitteen saaminen

```
String url = "rmi://example.com/Aurinkokunta";
AurinkokuntaInterface aurinkokuntaInterface
    = (AurinkokuntaInterface)Naming.lookup(url);
```

Rmiregistry toimii olioviitteiden välityspalveluna, jonka avulla asiakasohjelmat pystyvät nimen perusteella löytämään käytettävän olion. Nämä tiedot puolestaan Rmiregistry saa palvelinohjelmalta, joka välittää ne sille `java.rmi.naming`-luokan metodilla `rebind` [Schildt 2011, RMI]. Tästä esimerkki listauksessa 7.4.

Listaus 7.4: Olioviitteen rekisteröinti RMIRegistryyn

```
AurinkokuntaToteutus aurinkokuntaToteutus = new
    AurinkokuntaToteutus();
Naming.rebind("Aurinkokunta", aurinkokuntaToteutus);
```

Esimerkin tapauksessa olisi palvelinohjelma luonut instanssin aurinkokunnan tietoja tarjoavasta paikallisoliosta ja olisi rekisteröinyt sen löydettäväksi ”Aurinkokunta”-nimen alle Rmiregistryyn. Kun Rmiregistryn kautta on ensin löydetty ensimmäinen olio, voidaan sen kautta useissa tapauksissa löytää loputkin kutsuttavat metodit [rmiregistry]. AurinkokuntaToteutus-luokan lisäksi tarvitaan myös rajapinnan määritelmä, joka kertoo millaisia metodeja toteutuksessa on [Schildt 2011, RMI].

7.3.3 RMI-protokolla

RMI toimii ainoastaan, kun kumpikin osapuoli käyttää Javaa [RMI and RMI-IIOP]. Se ei siis sovellu eri ohjelmointikielillä toteutettujen ratkaisujen väliseksi kommunikointitavaksi. RMI-IIOP on RMI:n laajennos, joka parantaa tilannetta mahdollistamalla RMI-olioiden kommunikoinnin CORBA-olioiden kanssa. CORBA on standardi, jonka avulla eri ohjelmointikielillä toteutetut oliot voivat kutsua toistensa metodeja [Wikipedia:CORBA].

RMI:n taustalla on TCP/IP:n yllä toimiva protokolla JRMP (Java Remote Method Protocol). Protokollan ymmärtäminen ei ole välttämätöntä RMI:n käyttämiseksi, mutta on kuitenkin hyvä tiedostaa, että RMI:n toteutuksesta riippuen saattaa jokaista RMI-kutsua varten syntyä uusi sovikeyhteys. Koska jokainen TCP-yhteys vaatii uuden kättelyn, voisi tämä hidastaa toimintaa huomattavasti. Latenssin vähentämiseksi joissakin toteutuksissa vanhat yhteydet pysyvät tallessa, jotta uusi kutsu voidaan suorittaa viiveettä.

8 YHTEYDET ADOBE FLASH -SOVELMISTA

Flash on selaimen lisäosa, joka tunnettiin pitkään nimellä ”Macromedia Flash”. Adobe osti Macromedian vuonna 2005. Yrityksoston jälkeen nimi muuttui muotoon ”Adobe Flash”. Flash käyttää ActionScript-ohjelmointikieltä, joka on ECMAScript-standardikielen murrekieli.

8.1 Flashin loadVariables-funktio ja URLLoader-luokka

Luvussa 3.2.1 mainittiin lomakkeen lähettämisen yhteydessä lomakekenttien siirtomuoto ”application/x-www-form-urlencoded”. Flash mahdollistaa saman siirtomuodon käyttämistä päinvastaiseen suuntaan palvelimelta Flash-lisäosalle. Vielä ActionScriptin versiossa 2 käytetty funktio oli nimeltään loadVariables [loadVariables], mutta versiossa 3 toiminnallisuus siirtyi flash.net-paketin URLLoader-luokkaan. Esimerkkinä listauksessa 8.1 näkyy palvelimelle talletetun muuttujat.txt-tiedoston sisältö.

Listaus 8.1: muuttujat.txt-tiedoston sisältö

```
alaheimo=kissat&suku=felis&laji=catus
```

Listaus 8.2: URLLoaderin käyttö

```
package {
    import flash.net.*;
    import flash.display.MovieClip;
    import flash.events.*;

    public class App extends MovieClip {
        public function App() {
            var loader = new URLLoader();
            loader.dataFormat = URLLoaderDataFormat.VARIABLES;
            loader.addEventListener(Event.COMPLETE, function ()
            {
                trace("Alaheimo: " + loader.data.alaheimo);
            });
            loader.load(new URLRequest("http://localhost:8000/
            muuttujat.txt"));
        }
    }
}
```

URLLoaderin avulla voidaan tiedot hakea ja saada käyttöön kätevästi olion attribuutteina. Esimerkki uuden URLLoaderin käytöstä listauksessa 8.2. Aiem-

man ActionScript-version loadVariables-funktio toimi myös samaan tapaan, mutta vaati ennen huomattavasti vähemmän ohjelmakoodia. Nykyisin koodiin tuotavat kirjastot, sekä paketti- ja luokkarakenne pidentävät ohjelmakoodia.

Flash-lisäosalla voidaan hakea tietoa ainoastaan palvelimelta, jonka verkkonimi on täsmälleen sama kuin sen sivun verkkonimi, jolta Flash-ohjelma haettiin [loadVariables]. Jos on kuitenkin tarve pyytää tietoa muulta palvelimelta, ja kyseinen palvelin on myös ohjelman kehittäjän hallinnassa, voidaan tätä vaatimusta kiertää sijoittamalla palvelimen juurihakemistoon erityinen crossdomain.xml-tiedosto. Jos esimerkiksi palvelimella a.com sijaitsisi Flash-ohjelma, jonka on haettava tietoa palvelimelta b.com, on tämä mahdollista sallia sijoittamalla b-palvelimelle tiedosto <http://b.com/crossdomain.xml>. Flash-lisäosa tarkastaa tiedoston olemassaolon ennen tiedonsiirron sallimista.

8.2 Vastakkeet ActionScriptissä

Jos halutaan toteuttaa vaikkapa keskustelusovellus tai verkkopeli Flashilla, olisi URLLoader-luokkaa käyttäen tehtävä palvelimelle toistuvia HTTP-kutsuja, jotta uudet keskusteluviestit tai pelin siirrot saataisiin tietoon heti niiden tapahduttua. Jokainen kutsu vaatisi uuden TCP-yhteyden rakentamista, joka on hidas operaatio kolmitiekättelyn vuoksi [RFC793, 12]. Olisi parempi, jos yhteys voitaisiin luoda kerran ja pitää se auki. ActionScript 3.0:ssa on kaksi vaihtoehtoa jatkuvan yhteyden avaamiseksi palvelimeen kiertokyselyn sijaan: XMLSocket- ja Socket -luokat [Adobe.com:Socket Connections].

8.2.1 XMLSocket

XMLSocket-yhteyden yli siirretään XML-dokumentteja, jotka on eroteltu toisistaan nollatavulla [adobe.com:Socket Connections]. Listauksessa 8.3 on minimalistinen esimerkkiohjelma, joka avaa XMLSocket-yhteyden paikallispalvelimeen. Ohjelma lähettää palvelimelle testiviestin ja tulostaa vastaukset.

Listaus 8.3: XMLSocket-esimerkki

```

package {
    import flash.net.*;
    import flash.display.MovieClip;
    import flash.events.*;
    import flash.text.*;

    public class App extends MovieClip {
        public function App() {
            var socket = new XMLSocket();
            socket.addEventListener(Event.CONNECT, function (
                event) {
                    socket.send('<MESSAGE TEXT="Moro!"/>');
                });
            socket.addEventListener(DataEvent.DATA, function (
                event) {
                    trace(event.data); // XML-viesti saapui
                });
            socket.connect('localhost', 843);
        }
    }
}

```

Socket- ja XMLSocket-yhteydet ovat tiukemmassa kontrollissa kuin URLRequester-luokka. Oletuksena ei kumpikaan sovikeyhteys onnistu, vaikka palvelimen verkkonimi olisikin sama kuin se verkkonimi, jolta Flash-ohjelma ladattiin [Adobe.com>Loading data]. Selvittäääkseen ovatko sovikeyhteydet palvelimen haluttuun porttiin mahdollisia, selaimen Flash-lisäosa pyytää palvelimelta ”socket policy file” -XML-dokumentin. Dokumentin avulla palvelin voi kertoa Flash-lisäosalle hyväksyvänsä suorat sovikeyhteydet. Oletuksena Flash-lisäosa pyytää dokumenttia kutsulla porttiin 843. Myös policy-dokumentin siirto tapahtuu XMLSocket-protokollalla, joten tavallinen HTTP-palvelin ei riitä sen palvelemiseen. Protokolla on kuitenkin hyvin yksinkertainen: HTTP-pyynnön sijaan palvelin saa viestin ”<policy-file-request/>”, jonka jälkeen tulee nollatavu. Yhteyden onnistumiseksi viestiin on vastattava policy-dokumentilla, jonka jälkeen tulee myös nollatavu. Minimaalinen esimerkki listauksessa 8.4.

Listaus 8.4: Socket policy file

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "/xml/dtds/cross-domain-
  policy.dtd">
<cross-domain-policy>
  <site-control permitted-cross-domain-policies="master-only"/>
  <allow-access-from domain="*" to-ports="*" />
</cross-domain-policy>
```

8.2.2 Socket

Socket-luokan edustama binäärimuotoinen sovikeyhteys on kuten XMLSocket, mutta mahdollistaa vapaan datan siirron, ilman että sen tarvitsisi välttämättä olla XML-muotoista [adobe.com:Socket Connections]. Socket-luokka tuli uutena vaihtoehtona mukaan XMLSocketin rinnalle ActionScriptin versiossa 3. Listauksessa 8.5 esimerkki binäärimuotoisen sovikkeen avaamisesta.

Listaus 8.5: Binäärimuotoisen sovikkeen avaaminen

```
package {
    import flash.net.*;
    import flash.display.MovieClip;
    import flash.utils.ByteArray;

    public class App extends MovieClip {
        public function App() {
            var socket = new Socket();
            socket.connect('localhost', 843);
            var byteArray = new ByteArray();
            byteArray.writeUTF("Hei, maailma.");
            socket.writeBytes(byteArray);
            socket.flush();
        }
    }
}
```

8.3 Adobe Cirrus

Adobe Cirrus hoitaa Flash-maailmassa vastaavaa roolia, kuin luvussa 6.8 esiteltä WebRTC JavaScriptissä. Loppukäyttäjän verkkoyhteyttä suojaa yleensä palomuuuri. Välissä on yleensä myös IP-osoitteita muuntava NAT, jotka estävät käyttäjää ajamasta palvelinta. Eston vuoksi käyttäjät eivät siten voi avata so-

vikeyhteyttä suoraan toisilleen, vaikka se useassa tilanteessa voisi ratkaisuna olla parempi latenssin ja yksityisyydensuojan kannalta. Esimerkiksi videokeskustelusovelluksessa loppukäyttäjä lähettää oman videovirtansa palvelimelle, josta se edelleen lähetetään eteenpäin toiselle käyttäjälle. Suora yhteys vähentäisi turhaa kuormaa palvelimelle.

Aikaisemmalta nimeltään Stratus, Adobe Cirrus mahdollistaa palomuurien läpäisyn niin, että kaksi Flash-selainsovelmaa voivat viestiä ilman välityspalvelinta [Adobe:Cirrus]. Yhteyden muodostamiseen tarvitaan siis alussa palvelin, mutta yhteyden muodostuttua voivat käyttäjät kommunikoida suoraan toisilleen.

Kommunikointiin käytetty protokolla on RTMFP ("Real Time Media Flow Protocol"). Se perustuu UDP-datapaketteihin, jotta latenssi voidaan pitää mahdollisimman alhaisena [Adobe:Cirrus service]. TCP takaisi tietovirran eheyden, mutta reaaliaikaisissa palveluissa kuten puhelussa ja nopeatempoisissa peleissä käytetään UDP-paketteja, koska viiveen minimointi on tärkeämpää. Flashin mukana tulevat ääni- ja videokoodekit Speex ja H.264 kumpikin toimivat, vaikka datapaketteja puuttuisikin välistä, joten ne sopivat hyvin UDP:n kanssa käytettäväksi. Vaihtoehtona suoralle P2P-kommunikoinnille on välittää video palvelimen kautta. Tätä varten palvelimella on oltava ohjelmisto, joka ottaa mediavirran vastaan. Tällaisia palvelinohjelmistoja ovat Adobe Media Server, sekä sen avoimen lähdekoodin vaihtoehto Red5.

9 YHTEENVETO

Tämän tutkielman tarkoituksena oli esitellä kommunikointiin liittyvät selaimessa käytettävät tekniikat. Tutkielman aineisto koostui pääasiallisesti verkossa saatavilla olleista rajapintakuvauksista ja RFC-dokumenteista. Kuvailtujen tekniikoiden ohjelmakoodiesimerkkien toiminta testattiin paikallisissa testiympäristöissä.

Käyttäjien vähitellen päivittäessään selaimia uusimpiin versioihin riittänee tulevaisuudessa JavaScriptin comet-ohjelmointiin ainoastaan WebSockettien käyttö. WebSocket-tuen ollessa edelleen useissa suosituissa selainversioissa puutteellinen, suositellaan siirtymäaikana myös muiden comet-tekniikoiden tukemista. Tekniikoiden toimivuus vaihtelee suuresti selainversioiden välillä, joten suositellaan sellaisen valmiin kirjaston käyttöä, joka pystyy automaattisesti valitsemaan käyttäjän selaimessa toimivan kommunikointimenetelmän. Jos valmista kirjastoa ei haluta käyttää, näyttäisi WebSocket-tuen puuttuessa pitkäkiertokyselyihin tukeutuminen toteutukseltaan helpoimmalta vaihtoehdolta. Jos toteutukseen halutaan lisätä vielä kolmas tekniikka latenssin vähentämiseksi, kannattanee seuraavaksi lisätä tuki EventSource-rajapinnalle.

Vähälatenssisten pelien toteuttamiseen tarkastelluista tekniikoista suositellaan Javan tapauksessa DatagramSocket UDP-sovikkeiden käyttöä. Flashin ja JavaScriptin UDP-tuki on suunniteltu erityisesti P2P-videokonferenssitarkoituksiin, joten niiden käyttö palvelimen kanssa kommunikointiin on hankalampaa. Toteutus on helpompi, mikäli vastapelaajiin otetaan yhteys suoraan, sillä silloin P2P-rajapintojen tarjoamia vapaadatan siirtomahdollisuuksia voidaan käyttää suora- viivaisemmin. Tilanne kuitenkin monimutkaistuu, jos halutaan UDP-yhteys omalle palvelimelle, sillä tällöin palvelimen on toimittava P2P:n osapuolena.

Selaimessa toimivissa P2P-videokonferenssisovelluksissa on Flashin Adobe Cirrus ollut perinteisesti paras vaihtoehto, mutta WebRTC-rajapinnan myötä on videokeskustelun toteuttaminen JavaScriptilläkin tullut mahdolliseksi. Selainlisäosilla Flashilla ja Javallakin on kuitenkin edelleen asemansa, sillä niistä voidaan avata sellaisia suoria sovikeyhteyksiä, jotka saattaisivat JavaScriptin kautta vaatia välityspalvelinta. Kommunikointivaihtoehtojen lisääntymisen lisäksi myös JavaScriptin graafiset mahdollisuudet ovat parantuneet canvas- ja WebGL-tukien ansiosta. JavaScriptillä voidaankin nykyään toteuttaa entistä useampi aikaisemmin selainlisäosia vaatinut sovelma.

VIITELUETTELO

[Adobe.com>Loading data] *Loading data*. http://help.adobe.com/en_US/as3/dev/WS5b3ccc516d4fbf351e63e3d118a9b90204-7c60. (Tarkastettu 29.11.2013)

[Adobe.com>Socket Connections] *Socket Connections*.
http://help.adobe.com/en_US/ActionScript/3.0_ProgrammingAS3/WS5b3ccc516d4fbf351e63e3d118a9b90204-7cfb.html (Tarkastettu 26.10.2013)

[Adobe>Cirrus service] *Cirrus service for developing end-to-end applications using RTMFP in Flash Player 10*. Saatavana:
http://www.adobe.com/jp/devnet/flashplayer/articles/rtmfp_cirrus_app.html
(Tarkastettu 28.10.2013)

[Adobe>Cirrus] *Cirrus*. Saatavana: <http://labs.adobe.com/technologies/cirrus/>
(Tarkastettu 28.10.2013)

[as3corelib] *ActionScript 3.0 library for several basic utilities*. Saatavana:
<https://code.google.com/p/as3corelib/source/browse/trunk/src/com/adobe/serialization/json/JSONDecoder.as?r=83> (Tarkastettu 26.09.2013)

[Beams 2012] *Chris Beams ym., Introduction to WebSockets*. Saatavana:
<http://www.youtube.com/watch?v=z-CYO1ABCp4> (Tarkastettu 31.10.2013)

[Bug 843508 - Remove support for multipart XHR responses] *Bug 843508 - Remove support for multipart XHR responses*.
https://bugzilla.mozilla.org/show_bug.cgi?id=843508 (Tarkastettu 03.12.2013)

[Chase, 2004] *Nicholas Chase, Use a Java applet to access remote Web services*.
Saatavana: <http://www.ibm.com/developerworks/java/library/x-jappws/>
(Tarkastettu 27.09.2013).

[cl-json] *Json encoder and decoder for Common-Lisp*. Saatavana:
<https://github.com/hankhero/cl-json/blob/master/src/decoder.lisp> (Tarkastettu 26.09.2013)

- [Comet Programming] *Comet Programming: the Hidden IFrame Technique*. Saatavana: <http://www.webreference.com/programming/javascript/rg30/index.html> (Tarkastettu 25.10.2013)
- [COMET Streaming in Internet Explorer] *COMET Streaming in Internet Explorer*. <http://blogs.msdn.com/b/ieinternals/archive/2010/04/06/comet-streaming-in-internet-explorer-with-xmlhttprequest-and-xdomainrequest.aspx> (Tarkastettu 05.12.2013)
- [Comet: Low Latency Data for the Browser] *Comet: Low Latency Data for the Browser*. <http://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/> (Tarkastettu 02.12.2013)
- [Comet: Server-Push in Web] *Comet: Server-Push in Web*. <http://www.slideshare.net/mocheng/comet-server-pushoverweb> (Tarkastettu 29.11.2013)
- [Crockford] *JavaScript: The Good Parts*. O'Reilly Media, 2008.
- [ECMA-262] *ECMAScript Language Specification*. Saatavana: <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf> (Tarkastettu 15.10.2013)
- [Épardaud] *What every web developer must know about URL encoding*. <http://blog.lunatech.com/2009/02/03/what-every-web-developer-must-know-about-url-encoding>
- [FastCGI] *FastCGI Specification*. Saatavana: <http://www.fastcgi.com/devkit/doc/fcgi-spec.html> (Tarkastettu 10.10.2013)
- [fcgiapp] *fcgiapp*. Saatavana: <https://pypi.python.org/pypi/fcgiapp/1.4> (Tarkastettu 17.10.2013)
- [Getting Started with WebRTC] *Getting Started with WebRTC*. Saatavana: <http://www.html5rocks.com/en/tutorials/webrtc/basics> (Tarkastettu 06.11.2013)
- [Google I/O 2013] *Real-time communication with WebRTC: Google I/O 2013*. Saatavana: <http://www.youtube.com/watch?v=p2HzZkd2A40> (Tarkastettu 07.10.2013)

- [Harold, 2004] *Elliotte Rusty Harold, Java Network Programming*. O'Reilly, 2004.
- [Howto WSGI] *Howto WSGI*. Saatavana: http://redmine.lighttpd.net/projects/4/wiki/Howto_WSGI (Tarkastettu 17.10.2013)
- [HTML4.01 forms] *HTML 4.01 Specification*. Saatavana: <http://www.w3.org/TR/html401/interact/forms.html> (Tarkastettu 18.07.2013)
- [HTML5.1] *HTML5.1*. Saatavana: <http://www.w3.org/html/wg/drafts/html/CR/forms.html#url-encoded-form-data> (Tarkastettu 12.12.2013)
- [HTML5Nightly] *HTML 5.1 Nightly, A vocabulary and associated APIs for HTML and XHTML*. Saatavana: <http://www.w3.org/html/wg/drafts/html/master/browsers.html#origin> (Tarkastettu 26.09.2013)
- [HTTP/0.9] *The Original HTTP as defined in 1991*. Saatavana: <http://www.w3.org/Protocols/HTTP/AsImplemented.html> (Tarkastettu 21.05.2013)
- [HTTP1994] *Basic HTTP as defined in 1992*. Saatavana: <http://www.w3.org/Protocols/HTTP/HTTP2.html> (Tarkastettu 22.05.2013)
- [Huang] *Lin-Shung Huang ym., Talking to Yourself for Fun and Profit*. Saatavana: <http://w2spconf.com/2011/papers/websocket.pdf> (Tarkastettu 30.10.2013)
- [Implementing Script Tag Long Polling for Comet Applications] *Implementing Script Tag Long Polling for Comet Applications*. <http://carloscarrasco.com/articles/implementing-script-tag-long-polling-for-comet-applications/> (Tarkastettu 04.12.2013)
- [InfoWorld] *JavaScript creator ponders past, future*. Saatavana: <http://www.infoworld.com/d/developer-world/javascript-creator-ponders-past-future-704> (Tarkastettu 15.10.2013)
- [JSON.org] *JSON*. Saatavana: <http://www.json.org> (Tarkastettu 14.05.2013)
- [Korpela] *Mitä RFC:t ovat*. Saatavana: <http://www.cs.tut.fi/~jkorpela/rfc.html> (Tarkastettu 21.05.2013)

[Lighttpd] *Andre Bogus, Lighttpd*. Packt Publishing, 2008.

[Lintula 2004] *Petri Lintula, Suoraviestintää ja läsnäoloa SIP:illä*. Saatavana: http://www.cs.uta.fi/research/thesis/masters/Lintula_Petri.pdf (Tarkastettu 24.10.2013)

[loadVariables] *loadVariables* *function*. Saatavana: http://help.adobe.com/en_US/AS2LCR/Flash_10.0/help.html?content=00000575.html (Tarkastettu 24.07.2013)

[Luotonen 1998] *Tunneling TCP based protocols through Web proxy servers*. Saatavana: <http://tools.ietf.org/html/draft-luotonen-web-proxy-tunneling-01> (Tarkastettu 01.07.2013)

[Mozilla HTTP] *HTTP*. Saatavana: <https://developer.mozilla.org/en-US/docs/HTTP> (Tarkastettu 21.05.2013)

[Netstrings] *Netstrings*. Saatavana: <http://cr.yip.to/proto/netstrings.txt> (Tarkastettu 09.12.2013)

[Nginx third party modules] *Third party modules*. Saatavana: <http://wiki.nginx.org/3rdPartyModules> (Tarkastettu 15.10.2013).

[PEP3333] *Python Web Server Gateway Interface v1.0.1*. <http://www.python.org/dev/peps/pep-3333/> (Tarkastettu 18.10.2013)

[protobuf] *protobuf*. Saatavana: <https://code.google.com/p/protobuf/> (Tarkastettu 27.05.2013)

[Protocol Buffer Basics] *Protocol Buffer Basics: Java*. Saatavana: <https://developers.google.com/protocol-buffers/docs/javatutorial?hl=fr> (Tarkastettu 01.10.2013)

[Protocol Buffers Developer Guide] *Protocol Buffers Developer Guide*. Saatavana: <https://developers.google.com/protocol-buffers/docs/overview#whynotxml> (Tarkastettu 27.05.2013)

[protojs] *protojs*. Saatavana: <https://github.com/sirikata/protojs> (Tarkastettu 27.05.2013)

- [RFC1738] *Uniform Resource Locators (URL)*. Saatavana:
<http://www.ietf.org/rfc/rfc1738.txt> (Tarkastettu 30.05.2013)
- [RFC1867] *Form-based File Upload in HTML*. Saatavana:
<http://www.ietf.org/rfc/rfc1867.txt> (Tarkastettu 18.07.2013)
- [RFC1945] *Hypertext Transfer Protocol – HTTP/1.0*. Saatavana:
<http://www.ietf.org/rfc/rfc1945.txt> (Tarkastettu 29.05.2013)
- [RFC2068] *Hypertext Transfer Protocol – HTTP/1.1*. Saatavana:
<http://www.ietf.org/rfc/rfc2068.txt> (Tarkastettu 29.05.2013)
- [RFC2616] *Hypertext Transfer Protocol – HTTP/1.1*. Saatavana:
<http://www.ietf.org/rfc/rfc2616.txt> (Tarkastettu 21.05.2013)
- [RFC3875] *The Common Gateway Interface (CGI) Version 1.1*. Saatavana:
<http://tools.ietf.org/html/rfc3875> (Tarkastettu 10.10.2013)
- [RFC3986] *Uniform Resource Identifier (URI): Generic Syntax*. Saatavana:
<http://tools.ietf.org/html/rfc3986> (Tarkastettu 10.06.2013)
- [RFC4627] *The application/json Media Type for JavaScript Object Notation (JSON)*. Saatavana: <http://tools.ietf.org/html/rfc4627> (Tarkastettu 14.05.2013)
- [RFC5245] *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols*. Saatavana:
<http://tools.ietf.org/html/rfc5245> (Tarkastettu 19.11.2013)
- [RFC6455] *The WebSocket Protocol*. Saatavana:
<http://tools.ietf.org/html/rfc6455> (Tarkastettu 22.10.2013)
- [RFC793] *Transmission Control Protocol*. Saatavana:
<http://www.ietf.org/rfc/rfc793.txt> (Tarkastettu 12.12.2013)
- [RFC882] *Domain Names – Concepts and Facilities*. Saatavana:
<http://www.ietf.org/rfc/rfc882.txt> (Tarkastettu 09.06.2013)

- [RMI and RMI-IIOP] *IBM User Guide for Java V7 on z/OS: RMI and RMI-IIOP*. Saatavana: <http://pic.dhe.ibm.com/infocenter/java7sdk/v7r0/index.jsp?topic=>
- [rmiregistry] *rmiregistry - The Java Remote Object Registry*. Saatavana: <http://docs.oracle.com/javase/7/docs/technotes/tools/solaris/rmiregistry.html> (Tarkastettu 02.10.2013)
- [SCGI] *SCGI: A Simple Common Gateway Interface alternative*. Saatavana: <http://python.ca/scgi/protocol.txt> (Tarkastettu 21.10.2013)
- [Schildt 2011] *Herbert Schildt, Java The Complete Reference, 8th Edition*. McGraw-Hill Osborne Media, 2011.
- [Server-Sent Events] *Server-Sent Events*. <http://www.w3.org/TR/eventsource/> (Tarkastettu 06.12.2013)
- [simplejson] *Simplejson is a simple, fast, extensible JSON encoder/decoder for Python*. Saatavana: <https://github.com/simplejson/simplejson/blob/master/simplejson/decoder.py> (Tarkastettu 26.09.2013)
- [sock.js] *WebSocket emulation - Javascript client*. Saatavana: <http://sockjs.org> (Tarkastettu 31.10.2013)
- [socket.io] *Socket.IO: the cross-browser WebSocket for realtime apps*. Saatavana: <http://socket.io/> (Tarkastettu 31.10.2013)
- [Sormunen and Pennanen, 2004] Eero Sormunen and Sami Pennanen, *The challenge of automated tutoring in Web-based learning environments for information retrieval instruction*. Information Research 9 2 (2004). Saatavana: <http://informationr.net/ir/9-2/paper169.html>
- [Sormunen and Pennanen, 2004] Eero Sormunen and Sami Pennanen, *The challenge of automated tutoring in Web-based learning environments for information retrieval instruction*. Information Research 9 2 (2004). Saatavana: <http://informationr.net/ir/9-2/paper169.html> (Tarkastettu 12.12.2013)

- [The Future of Comet] *The Future of Comet: Part 1, Comet Today.*
<http://cometdaily.com/2007/12/11/the-future-of-comet-part-1-comet-today/>
 (Tarkastettu 05.12.2013)
- [tutorialpoint] *JDBC - Sample, Example Code.* Saatavana:
<http://www.tutorialspoint.com/jdbc/jdbc-sample-code.htm> (Tarkastettu
 30.09.2013)
- [W3 SOAP] *Simple Object Access Protocol (SOAP) 1.1.*
<http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>. Saatavana: (Tar-
 kastettu 14.05.2013)
- [W3C Mailing List] *<keygen> element* [http://lists.w3.org/Archives/Public/public-
 html/2009Sep/0043.html](http://lists.w3.org/Archives/Public/public-html/2009Sep/0043.html) (Tarkastettu 04.10.2013)
- [WebRTC 1.0] *WebRTC 1.0: Real-time Communication Between Browsers.*
<http://dev.w3.org/2011/webrtc/editor/webrtc.html> (Tarkastettu 07.10.2013)
- [WebRTC Overview] *WebRTC: An Overview.* Saatavana:
<http://proness.kix.in/talks/fluent13-webrtc> (Tarkastettu 02.10.2013)
- [WebSocket API] *The WebSocket API.* Saatavana:
<http://www.w3.org/TR/websockets/> (Tarkastettu 05.11.2013)
- [websocket.org] *About HTML5 WebSockets.* Saatavana:
<http://www.websocket.org/aboutwebsocket.html> (Tarkastettu 23.10.2013)
- [Wikipedia:Berkeley Sockets] *Berkeley Sockets.* Saatavana:
http://en.wikipedia.org/wiki/Berkeley_sockets (Tarkastettu 11.10.2013)
- [Wikipedia:Common Gateway Interface] *Common Gateway Interface.* Saatava-
 na: http://en.wikipedia.org/wiki/Common_Gateway_Interface (Tarkastettu
 11.10.2013)
- [Wikipedia:CORBA] *Common Object Request Broker Architecture.* Saatavana:
http://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture
 (Tarkastettu 28.10.2013)
- [Wikipedia:HTTP] *HTTP.* Saatavana: <http://en.wikipedia.org/wiki/Http> (Tar-
 kastettu 22.05.2013)

[Wikipedia:JSON-WSP] *JSON-WSP*. <http://en.wikipedia.org/wiki/JSON-WSP>. Saatavana: (Tarkastettu 14.05.2013)

[Wikipedia:JSON] *JSON*. Saatavana: <http://en.wikipedia.org/wiki/JSON> (Tarkastettu 15.05.2013)

[Wikipedia:JVM languages] *List of JVM languages*. Saatavana: http://en.wikipedia.org/wiki/List_of_JVM_languages (Tarkastettu 19.08.2013).

[Wikipedia:Out-of-band] *Out-of-band*. Saatavana: <http://en.wikipedia.org/wiki/Out-of-band> (Tarkastettu 25.10.2013)

[Wikipedia:RMI] *Java remote method invocation*. Saatavana: http://en.wikipedia.org/wiki/Java_remote_method_invocation (Tarkastettu 27.09.2013)

[XMLSocket] *flash.net.XMLSocket - AS3*. Saatavana: http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/net/XMLSocket.html (Tarkastettu 25.10.2013) #Thereservedcharactersaredifferentforeachpart (Tarkastettu 21.06.2013)

10 LYHENTEET

AJAX	Asynchronous JavaScript and XML (Ajax)
API	Application Programming Interface
BSD	Berkeley Software Distribution
CGI	Common Gateway Interface
CRUD	Create Read Update Delete
DNS	Domain Name System
DOM	Document Object Model
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
IP	Internet Protocol
JDBC	Java Database Connectivity
JRMP	Java Remote Method Protocol
JSON	JavaScript Object Notation
PEP	Python Enhancement Proposal
PHP	PHP: Hypertext Preprocessor
RFC	Request For Comments
RMI	Remote Method Invocation
SCGI	Simple Common Gateway Interface
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
URL	Uniform Request Locator
UTF	UCS Transformation Format
WebRTC	Web Real Time Communication
WSGI	Python Web Server Gateway Interface
XML	Extensible Markup Language