

hyväksymispäivä

arvosana

arvostelija

## **Hajautettu sisällönjakelu videonjakopalvelussa**

Ilari Moilanen

Helsinki 27.05.2016

HELSINGIN YLIOPISTO  
Tietojenkäsittelytiede

Tiedekunta – Fakultet – Faculty		Laitos – Institution – Department	
Matemaattis-luonnontieteellinen tiedekunta		Tietojenkäsittelytieteen laitos	
Tekijä – Författare – Author			
Ilari Moilanen			
Työn nimi – Arbetets titel – Title			
Hajautettu sisällönjakelu videonjakopalvelussa			
Oppiaine – Läroämne – Subject			
Tietojenkäsittelytiede			
Työn laji – Arbetets art – Level	Aika – Datum – Month and year	Sivumäärä – Sidoantal – Number of pages	
Pro gradu -tutkielma	27.05.2016	50	
Tiivistelmä – Referat – Abstract			
<p>Ihmisten käyttäytyminen internetissä voi määräytyä hyvin sattumanvaraisesti. Esimerkiksi sosiaalisen median kautta uudet ja kiinnostavat asiat voivat levitä hyvinkin nopeasti usealle eri ihmiselle. Jos sisällönjakaja ei ole varautunut, tai pystynyt varautumaan esimerkiksi rahallisista syistä, piikkeihin kysynnässä, niin odottamaton kiinnostus voi vaikuttaa palvelun laatuun tai jopa estää palvelun kokonaan.</p> <p>Esimerkkinä tilanteesta, jossa sisällön jakaja ei pysty täysin varautumaan äkkinäiseen resurssitarpeeseen, otetaan pieni videonjakopalvelu. Palvelua käyttää pieni vakiokäyttäjien joukko, jonka palveluun tarvittavan resurssimäärän yksittäinen palvelin pystyy hyvin tarjoamaan. Jos tällaisessa tilanteessa yksittäinen video menee viraaliksi sosiaalisessa mediassa ja käyttäjäkunta nousee hetkellisesti siihen astisesta maksimaalisesta ruuhkahuipusta 10-1000 – kertaiseksi, voi palvelin kaatua kokonaan tai ainakaan se ei pysty tarjoamaan ainoallekaan käyttäjälle palvelua.</p> <p>Jopa perinteinen sisällönjakeluverkko (CDN, content distribution network) auttaa huonosti tilanteeseen, jossa syntynyt salamajoukko (flash crowd) on vahvasti paikkaan sidottu ilmiö. Sisällönjakeluverkossa tieto jakaantuu kyllä suuremmalle maantieteelliselle alueelle kuin vain yksittäiseen keskuskoneeseen, mutta kun tarvitaan paljon kapasiteettia pienemmässä, vahvasti lokaalissa, paikallisessa osassa ei sisällönjakeluverkko olekaan enää paras ratkaisu ongelmaan.</p> <p>Tässä pro gradu -tutkielmassa keskitytään tutkimaan miten hyvin rajatuilla resursseilla varustettu palveluntarjoaja pystyisi rakentamaan suorituskykyisen videonjakopalvelun hajautetulla sisällönjakelulla. Samalla varmistetaan, että mahdollisen salamajoukon aiheuttaman käyttäjäpiikin aiheuttama ongelma ei johda palvelun kaatumiseen. Ratkaisuvaihtoehdoissa keskitytään käyttäjille hajautettuun ratkaisuun, koska palvelinkapasiteettia nimenomaan ei haluta lisätä.</p> <p>Jos pienemmillä resursseilla varustetut yksityishenkilöt voisivat ylläpitää omia palveluitaan halvalla ratkaisulla, saavutettaisiin monia etuja. Näitä etuja ovat muun muassa mahdollisen sensuurin väheneminen, yksityisyyden suoja, kilpailun mahdollistaminen, sopeutumiskyky ja virheiden sietokyky.</p>			
ACM Computing Classification System (CCS 2012):			
<ul style="list-style-type: none"> <li>• <b>Networks</b> → <b>Peer-to-peer protocols</b></li> <li>• <b>Software and its engineering</b> → <b>Peer-to-peer architectures</b></li> <li>• Networks → Network measurement</li> </ul>			
Avainsanat – Nyckelord – Keywords			
bittorrent, hajautus, WebRTC, P2P, HTML5			
Säilytyspaikka – Förvaringställe – Where deposited			
Muita tietoja – Övriga uppgifter – Additional information			

1	Johdanto	1
1.1	Tutkielman aihe	2
1.2	Rakenne	2
2	Teknologia	2
2.1	Asiakas-palvelin -malli	2
2.2	Vertaisverkko	3
2.3	Websocket	3
2.4	Sisällönjakeluverkko	3
2.4.1	Vertaisavusteinen sisällönjakelu	3
2.4.2	Mukautuva sisällönjakelu	4
2.5	Saman alkuperän käytäntö	4
2.6	WebRTC	4
2.7	BitTorrent	6
2.7.1	Torrent tiedosto	7
2.7.2	Asiakas	7
2.7.3	Seurantapalvelin	7
2.7.4	Parvi	9
2.7.5	Hajautettu tiivistetaulu	10
2.8	Suoratoisto	11
2.9	Selaimen lisäosa	12
2.10	HTML5	13
2.11	Media Source Extensions	13
2.11.1	Mukautuva videovuo	13
2.11.2	Aikasiirtymät	13
2.11.3	Yleistä	14
2.12	Java-sovelma	14
2.13	Java Web Start	14
2.14	NAT	15
2.15	ICE	15
2.15.1	STUN	15
2.15.2	TURN	16
3	Mahdolliset lähestymistavat	16
3.1	Kriteeristö	16
3.2	HTML5 sovellettuna	17
3.3	Selaimen lisäosa	18
3.4	Java-sovelma	19
3.5	Java Web Start	20
3.6	WebRTC	21

3.7	WebRTC ja BitTorrent	22
3.8	Vertailua ja valinta	23
3.8.1	Vertailun läpikäynti	23
3.8.2	Valinta	25
4	Koodaus	26
4.1	Koodaus	26
4.2	Koodin kuvaus	27
4.2.1	WebRTC ja BitTorrent	28
4.2.2	Videon toisto	29
4.2.3	Esimerkit	29
4.2.4	Ongelmat	29
5	Testaus	30
5.1	Testauksen toteutus	30
5.1.1	Mitä testataan	30
5.1.2	Testattava järjestelmä	30
5.1.3	Skenaariot	32
5.2	Tulokset	37
5.2.1	Tarkempaa tarkastelua (Skenaario 1)	39
5.2.2	Tarkempaa tulosten tulkintaa	41
5.3	Yhteenveto	45
6	Yhteenveto	46
6.1	Loppupäätelmät	47
6.2	Jatkotutkimuskohteita	47
	Lähdeluettelo	48

Liite 1. Koodiesimerkki

# 1 Johdanto

Ihmisten käyttäytyminen internetissä voi määräytyä hyvin sattumanvaraisesti. Esimerkiksi sosiaalisen median kautta uudet ja kiinnostavat asiat voivat levitä hyvinkin nopeasti usealle eri ihmiselle. Nämä kiinnostavat asiat eivät ole pelkkää tekstisisältöä vaan teknologin kehitys on tuonut yhä laajemman kirjon erilaisia sisältömuotoja, joiden jakaminen vaatii paljon resursseja. Jos resurssin jakajalla on resursseja jo entuudestaan paljon (esimerkiksi uutissivustot), eivät hetkelliset piikit kysynnässä välttämättä näy palvelun laadussa. Jos taas sisällönjakaja ei ole varautunut, tai pystynyt varautumaan esimerkiksi rahallisista syistä, piikkeihin, niin odottamaton kiinnostus voi vaikuttaa palvelun laatuun tai jopa estää palvelun kokonaan.

Esimerkkinä tilanteesta, jossa sisällön jakaja ei pysty täysin varautumaan äkkinäiseen resurssitarpeeseen, otetaan pieni videonjakopalvelu. Palvelulla on kenties vain yksi palvelinkone käytössään, joten palvelintason skaalautuminen ei onnistu. Palvelua käyttää pieni vakiokäyttäjien joukko, jonka palveluun tarvittavan resurssimäärän yksittäinen palvelin pystyy hyvin tarjoamaan. Jos tällaisessa tilanteessa yksittäinen video menee viraaliksi sosiaalisessa mediassa ja käyttäjäkunta nousee hetkellisesti siihen astisesta maksimaalisesta ruuhkauihasta 10-1000 –kertaiseksi, voi palvelin kaatua kokonaan tai ainakaan se ei pysty tarjoamaan ainoallekaan käyttäjälle palvelua.

Toisena esimerkkinä Wikipedia säätiö on herännyt tilanteeseen, jossa teksti- ja kuvasisältö vie huomattavasti vähemmän siirtokaistaa ja tallennustilaa kuin video ja audiotiedostot [BPD10]. Tässä tapauksessa tarjoajalla on suuret määrät palvelinresursseja mutta suhteessa vielä suurempi määrä käyttäjiä ja sisältöä jaettavanaan.

Jopa perinteinen sisällönjakeluverkko (CDN, content distribution network) auttaa huomasti tilanteeseen, jossa syntynyt salamajoukko (flash crowd) on vahvasti paikkaan sidottu ilmiö. Sisällönjakeluverkossa tieto jakaantuu kyllä suuremmalle maantieteelliselle alueelle kuin vain yksittäiseen keskuskoneeseen, mutta kun tarvitaan paljon kapasiteettia pienemmässä, vahvasti lokaalissa, paikallisessa osassa ei sisällönjakeluverkko olekaan enää paras ratkaisu ongelmaan [ZZF12]. Jos esimerkiksi Suomessa yksittäinen video menee viraaliksi sosiaalisen median kautta, on hyvin todennäköistä, että kapasiteettia tarvitaan paljon mahdollisimman lähellä videon alkuperäistä lähdettä, kun taas vaikkapa Kiinassa ei videolla ole sen esilläolon aikana ainuttakaan näkijää.

### **1.1 Tutkielman aihe**

Tässä pro gradu -tutkielmassa keskitytään tutkimaan miten hyvin rajatuilla resursseilla varustettu palveluntarjoaja pystyisi rakentamaan suorituskykyisen videonjakopalvelun hajautetulla sisällönjakelulla. Samalla varmistetaan, että mahdollisen salamajoukon aiheuttaman käyttäjäpiikin aiheuttama ongelma ei johda palvelun kaatumiseen. Ratkaisuvaihtoehdoissa keskitytään käyttäjille hajautettuun ratkaisuun, koska palvelinkapasiteettia nimenomaan ei haluta lisätä [SRS02] [ZCB11]. Käyttäjille hajautetusta videonjakopalveluista on jo olemassa tutkimusta [CSD07]. Kun sisällön jakelusta vastaavat käyttäjät, saavutetaan myös tilanne, jossa käyttäjien määrän lisääntyminen voi jopa johtaa palvelun toimimiseen paremmin kuin pienellä määrällä käyttäjiä.

Jos pienemmillä resursseilla varustetut yksityishenkilöt voisivat ylläpitää omia palveluitaan halvalla ratkaisulla, saavutettaisiin monia etuja. Näitä etuja ovat muun muassa mahdollisen sensuurin väheneminen, yksityisyyden suoja, kilpailun mahdollistaminen, sopeutumiskyky ja virheiden sietokyky.

### **1.2 Rakenne**

Luvussa yksi kuvataan ongelma, jolle etsitään ratkaisua. Sen jälkeen luvussa kaksi käydään läpi erilaiset teknologiat, joita tässä tutkielmassa tulee vastaan. Luvussa kolme kuvataan erilaisia ratkaisuvaihtoehtoja ongelman ratkaisemiseksi. Eri ratkaisuvaihtoehtoja vertaillaan keskenään ja niistä valitaan paras toteutettavaksi. Luvussa neljä hahmotellaan toteutusta valitulle ratkaisuvaihtoehdolle. Toteutuksesta ei yritetä tehdä täysin valmista ratkaisua vaan sellainen, joka toimii rajatussa ympäristössä. Luvussa viisi testataan toteutusta ja käydään läpi testituloksia. Luku kuusi päättää tutkielman ja se sisältää lyhyen yhteenvedon ja ehdotuksia siitä, miten tämän tutkielman tuloksia voisi käyttää hyödyksi jatkossa.

## **2 Teknologia**

Tässä luvussa käydään läpi eri termit ja teknologiat, joita tässä tutkielmassa on käytetty.

### **2.1 Asiakas-palvelin -malli**

Asiakas-palvelin -malli on yleisesti käytetty ohjelmistoarkkitehtuuri [DJT96].

Asiakas-palvelin -mallissa tiedoston jako käyttäjille tapahtuu siten, että käyttäjät pyytä-

vät resurssia, esimerkiksi tiedostoa, palvelimelta ja palvelin toimittaa sen suoraan sitä pyytäneelle käyttäjälle.

## **2.2 Vertaisverkko**

Vertaisverkko (peer-to-peer, P2P) on hajautetun viestinnän malli, jossa kaikilla osapuolilla on samat ominaisuudet ja kuka tahansa osapuoli voi aloittaa viestinnän. Toisin kuin asiakas-palvelin -mallissa vertaisverkko mahdollistaa kunkin solmun toimimisen sekä asiakkaana että palvelimena. Verkossa ei ole kiinteitä jäseniä vaan siihen voi saapua tai siitä voi poistua jäseniä milloin vain. Verkon toiminnot, kuten tiedostojen etsintä tai siirto, ovat verkon jäsenien tuottamia palveluita itselleen. Verkon toiminnot eivät näin ollen tule mistään keskitetystä lähteestä.

## **2.3 Websocket**

Normaalissa TCP-yhteydessä asiakas pyytää palvelimelta palvelua ja palvelin vastaa toimittaen halutun palvelun. Yhteyden aikana sisältödataa siirtyy vain jompaankumpaan suuntaan ja lopuksi yhteys suljetaan. Websocket-protokolla mahdollistaa saman TCP-yhteyden käyttämisen tiedonsiirtoon molempiin suuntiin. Koska websocketiin liittyvät käyttötapaukset ovat luonteeltaan pitkäaikaisempia, on myös luontevaa, että yhteyttä ei suljeta tiedonsiirtojen välillä. Yksi tyypillisimpiä esimerkkejä websocketin käytöstä ovat ns. chattipalvelut, joissa uusia viestejä liikkuu jatkuvasti asiakkailta palvelimelle ja palvelimelta taas chatin muille jäsenille.

## **2.4 Sisällönjakeluverkko**

Perinteisessä asiakas-palvelin -mallissa sisältö jaetaan verkon ylitse yhdeltä palvelimelta asiakkaalle. Sisällönjakelu voi olla myös hajautettuna verkoksi (sisällönjakeluverkko, content delivery network), jolloin sisältö ei tulekaan aina vain yhdeltä tietyltä palvelimelta, vaan mahdollisesti vaihtuvasta lähteestä tai jopa useasta eri lähteestä.

### **2.4.1 Vertaisavusteinen sisällönjakelu**

Vertaisavusteisessa sisällönjakelussa (Peer assisted content delivery) myös asiakkaat ottavat osaa sisällön jakamiseen [ZAC13] [ALR10]. Tällaisessa järjestelmässä on yleensä edelleen palvelin yhtenä osapuolena tarjoamassa sisällön (esimerkiksi videotiedoston) ensimmäisen kerran verkkoon, mutta sen jälkeen asiakkaat osallistuvat sen jakami-

seen jossain määrin [DKP10].

## 2.4.2 Mukautuva sisällönjakelu

Mukautuvassa sisällönjakelussa (Adaptive content delivery) on kyse siitä, että tilanteesta riippuen samaa sisältöä jaetaan asiakkaalle eri tavoin [JaM04]. Jos esimerkiksi käyttäjien määrä on pieni, voidaan sisältöä jakaa yksittäiseltä palvelimelta suoraan. Kun käyttäjien määrä kasvaa suuremmaksi siirrytäänkin käyttämään jotain muuta tapaa jakaa sisältöä. Sisällönjakeluverkon ylläpitäjä on voinut esimerkiksi päättää, että sisällön jo itselleen ladanneet käyttäjät jakavat sitä eteenpäin vertaisavusteisen sisällönjakelun keinoin.

## 2.5 *Saman alkuperän käytäntö*

Saman alkuperän käytäntö (same-origin policy) tarkoittaa sitä, että verkkosivustolla oleva skripti ei pysty käsittelemään dataa eri lähteestä, kuin mistä se itse on peräisin. Jos esimerkiksi verkkosivu on ladattu osoitteesta [esimerkki.fi](http://esimerkki.fi), ei kyseisellä verkkosivulla oleva skripti pysty pääsemään käsiksi verkkosivulta [toinenesimerkki.fi](http://toinenesimerkki.fi) ladattuun tietoon. Rajoitus on toteutettu jokaiseen Internet-selaimeen ja sitä ei voi ohittaa kuin hyvin rajatusti. Rajoitus estää esimerkiksi sen että pahantahtoisesti toteutettu sivusto ei pysty kaappaamaan käyttäjän istuntoa verkkopankkisivustolla.

## 2.6 *WebRTC*

WebRTC (Web Real-Time Communication, tosiaikainen kommunikointi verkossa) mahdollistaa kahden tai useamman käyttäjän välisen tiedonsiirron Internet-selaimien kautta. Jossain määrin mahdollista tämä on myös erinäisten selainten lisäosion avulla, mutta WebRTC toimii uusimmissa selaimissa sellaisenaan. Siirretty tieto voi olla periaatteessa minkälaista tahansa. Tuettuja tiedostomuotoja ovat teksti, binääridata ja datavirta (stream). Tekstin siirto onnistuu aina, mutta binääridataa siirrettäessä vastapuolen pitää tietää minkälaista dataa se on vastaanottamassa, pystyäkseen käsittelemään saamansa tiedon [Ris14]. Tiettyjä kuvan ja äänen tiedostomuotoja voi siirtää datavirtana. Datavirran tapauksessa ei siirrettävä tieto ole tiedosto vaan jatkuvaa virtaa videota ja/tai ääntä. Vastaanottaja toistaa saamaansa mediaa saman tien ja saatua dataa ei välttämättä tallenneta ollenkaan vastaanottopäässä. Datavirta onnistuu silloin, jos siirretyn tiedon tiedostomuoto on sellainen, että vastapuoli pystyy sitä toistamaan suoraan.



WebRTCn protokollamäärittelyä kehittää Internet Engineering Task Force (IETF) ja sen ohjelmointirajapintamäärittelyä puolestaan World Wide Web Consortium (W3C). Näiden kahden eri määrittelyn yhteisenä tarkoituksena on tuottaa standardi, jonka kaikki Internet-selaimet tulevaisuudessa toteuttaisivat. WebRTC on kokoelma erilaisia protokollia, kodekkeja ja mekanismeja. Ohjaavana periaatteena WebRTC projektilla on että sen ohjelmointirajapinta on open sourcea, ilmaista, standardoitua, Internet-selaimiin sisäänrakennettua ja tehokkaampi kuin olemassa olevat teknologiat [Alv16].

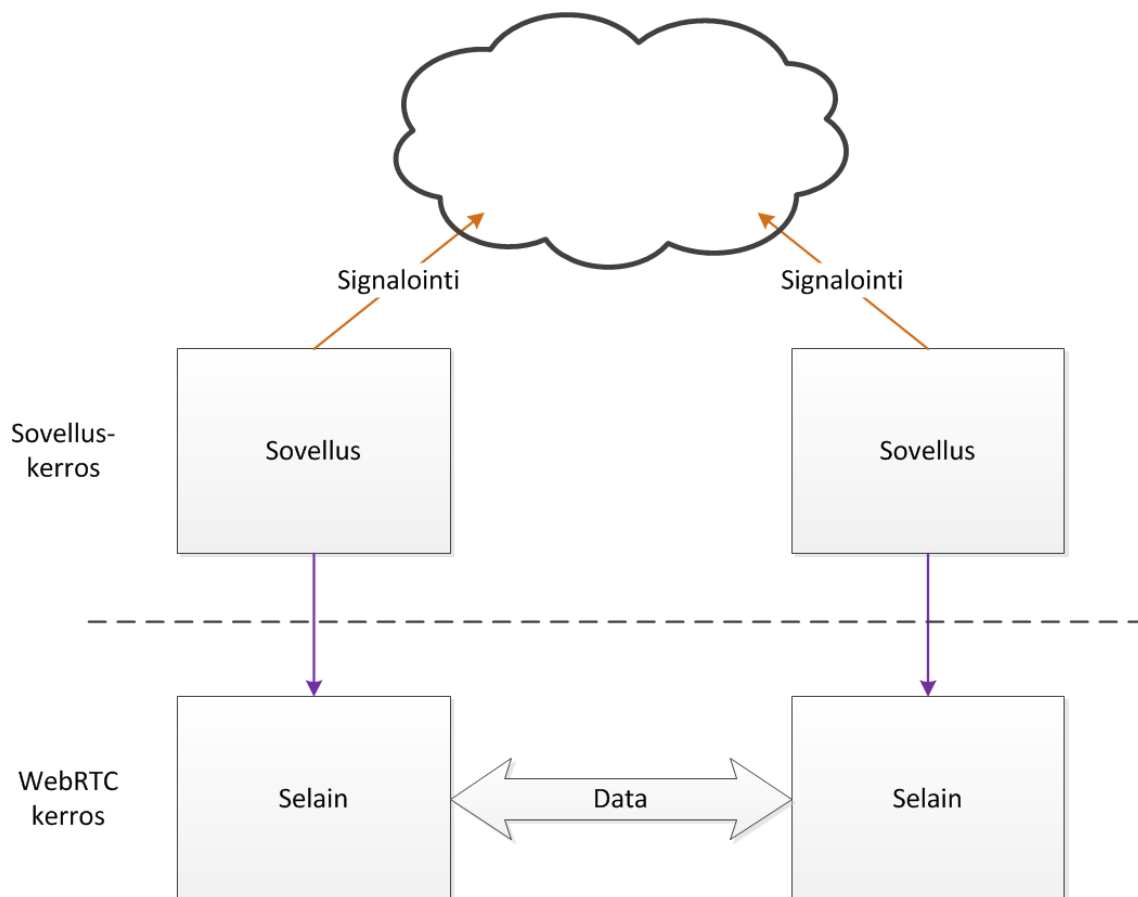
Vaikka tieto sinällään siirretään kahden tai useamman käyttäjän välillä, tarvitaan jokin mekanismi, jolla käyttäjät löytävät toisensa. Samoin tarvitaan joku tapa kertoa vastapuolelle kuka käyttäjistä on siirtämässä dataa, kenelle, missä muodossa, milloin jne. WebRTC ei itsessään standardoi tätä metatiedon siirtoa vaan jättää käytännön toteutuksen kulloisenkin sovelluksen omaksi huoleksi [Dut13]. Joka tapauksessa metatiedon välittämiseen tarvitaan palvelin, sillä tiedon välittämiseen tarvittavaa tietoa ei pysty välittämään ilman tiedon välittävää tietoa.

WebRTC käyttää termiä signaalointi (signaling) kun puhutaan kommunikaation koordinoinnista. Kommunikaatiossa tarvitaan periaatteessa kolmenlaisia palvelimia:

- sovellus täytyy pystyä tarjoamaan käyttäjälle eli esimerkiksi Internet-sivuston tarjoava palvelin
- signaaloinnin tarjoava palvelin
- reitityksen avuksi palvelin (jos jompikumpi käyttäjistä on NATin takana)

Kuvassa 1 on esitettynä WebRTC sovelluksen esimerkkirakenne. Esimerkkirakenne on tässä tapauksessa videopuhelusovelluksesta. Sovelluksen toiminta on seuraava:

- Sekä soittaja että puhelun vastaanottaja lataavat WebRTC sovelluksen (esimerkiksi Internet-sivulla olevan Javascript sovelluksen) selaimeensa
- Molemmat osapuolet käyttävät signaalointia löytääkseen aluksi toisensa ja sen jälkeen sopiaakseen videopuhelusession aloituksesta ja muista yleisistä asioista
- Signaalointivaiheessa ja mahdollisesti lisäksi itse datan siirrossa käytetään ICE metodeita (esimerkiksi STUN ja TURN palvelimet) (jätetty pois kuvasta yksinkertaistamisen vuoksi)
- Tämän jälkeen videopuhelu muodostetaan käyttäen WebRTC:tä



Kuva 1. WebRTC sovelluksen esimerkkirakenne

## 2.7 BitTorrent

BitTorrent on tiedostonjakoprotokolla joka tähtää sisällön tehokkaaseen monistamiseen verkon yli [BTS16] [KSK08]. Se on P2P (peer-to-peer, vertaisverkko) protokolla tarkoittaen että tiedosto jaetaan käyttäjiltä käyttäjille enemmän käytetyn asiakas-palvelin -mallin sijaan. Asiakas-palvelin -mallissa, jos käyttäjä haluaa jakaa sisältöä, hän lataa sen ensin palvelimelle, josta muut käyttäjät voivat sen sitten ladata itselleen. Tämän alustavan latauksen lisäksi palvelimen kautta kierrättämisessä on muitakin haittapuolia. Jos palvelimella on pieni kaistanleveys jompaankumpaan tai molempiin suuntiin voi tiedoston jakoon kuluva aika olla pitkä. Tämän lisäksi palvelin joutuu palvelemaan kaikkia käyttäjiä yhtäaikaan. P2P protokollissa jakautuu tiedonsiirto suuremmalle joukolla. Jakamisen nopeuttamiseksi entisestään jaettava tiedosto on jaettu pienempiin palasiin. Palakoko on tyypillisesti väliltä 256KT – 8MT ja jokainen palanen on edelleen jaettu pienempiin lohkoihin. Tällä tavoin tiedoston jako jakautuu palasten jakoon, jolloin tiedoston jako edelleen eteenpäin voi alkaa jo sillä hetkellä, kun alkuperäinen jakaja saa ensimmäisen palasen jaettua toiselle käyttäjälle. Jakoprosessin alussa tiedon liikkeen mää-

rä ei ole kovin suuri, mutta se saa lisää vauhtia ajan kuluessa, kun yhä useampi käyttäjä saa palasia ja jakaa niitä edelleen eteenpäin.

### 2.7.1 Torrent tiedosto

Torrent tiedosto sisältää metadataa BitTorrent-protokollalla jaettavasta sisällöstä. Torrent tiedostosta käyttäjä saa tarvitsemansa tiedot sisällön lataamisen tai jakamisen aloittamiseen.

Torrent tiedosto sisältää mm. seuraavat tiedot

- Jaettavan sisällön koostavien palasten koon ja lukumäärän. Tämä tiedon avulla BitTorrent-asiakasohjelma pystyy esimerkiksi varaamaan riittävän tilan tiedostojärjestelmästä sisällön lataamista varten.
- Jaettavan sisällön koostavan palasten tiivistet. Näiden avulla BitTorrent-asiakasohjelma pystyy palasen ladattuaan tekemään tarkistuslaskelman siitä, onko ladattu palanen vastaanotettu ehjänä.
- Jaettavan sisällön tiedostorakenteen. Jos sisältö koostuu esimerkiksi kansioista ja useammasta tiedostosta on BitTorrent-asiakasohjelman osattava koostaa vastaanotetusta tiedosta tuo samainen rakenne.
- Sisällön jaossa avustavien seurantapalvelinten yhteysosoitteet (eli niin sanotut Announce osoitteet). Näitä käyttäen BitTorrent-asiakasohjelma saa seurantapalvelimilta tiedon muista kyseisen sisällön jakamiseen osallistuvista asiakkaista.

### 2.7.2 Asiakas

Käyttäjä tarvitsee asiakasohjelman BitTorrent-protokollaa käyttääkseen. Perinteisesti asiakasohjelmana on toiminut erikseen ladattava työpöytäsovellus. Asiakasohjelma voi olla myös sulautettuna johonkin muuhun sovellukseen, kuten Internet-selaimeen.

BitTorrent-tiedostoa käyttäen asiakasohjelma osaa ottaa yhteyden muihin asiakasohjelmiin ja ladata tai jakaa haluttua sisältöä.

### 2.7.3 Seurantapalvelin

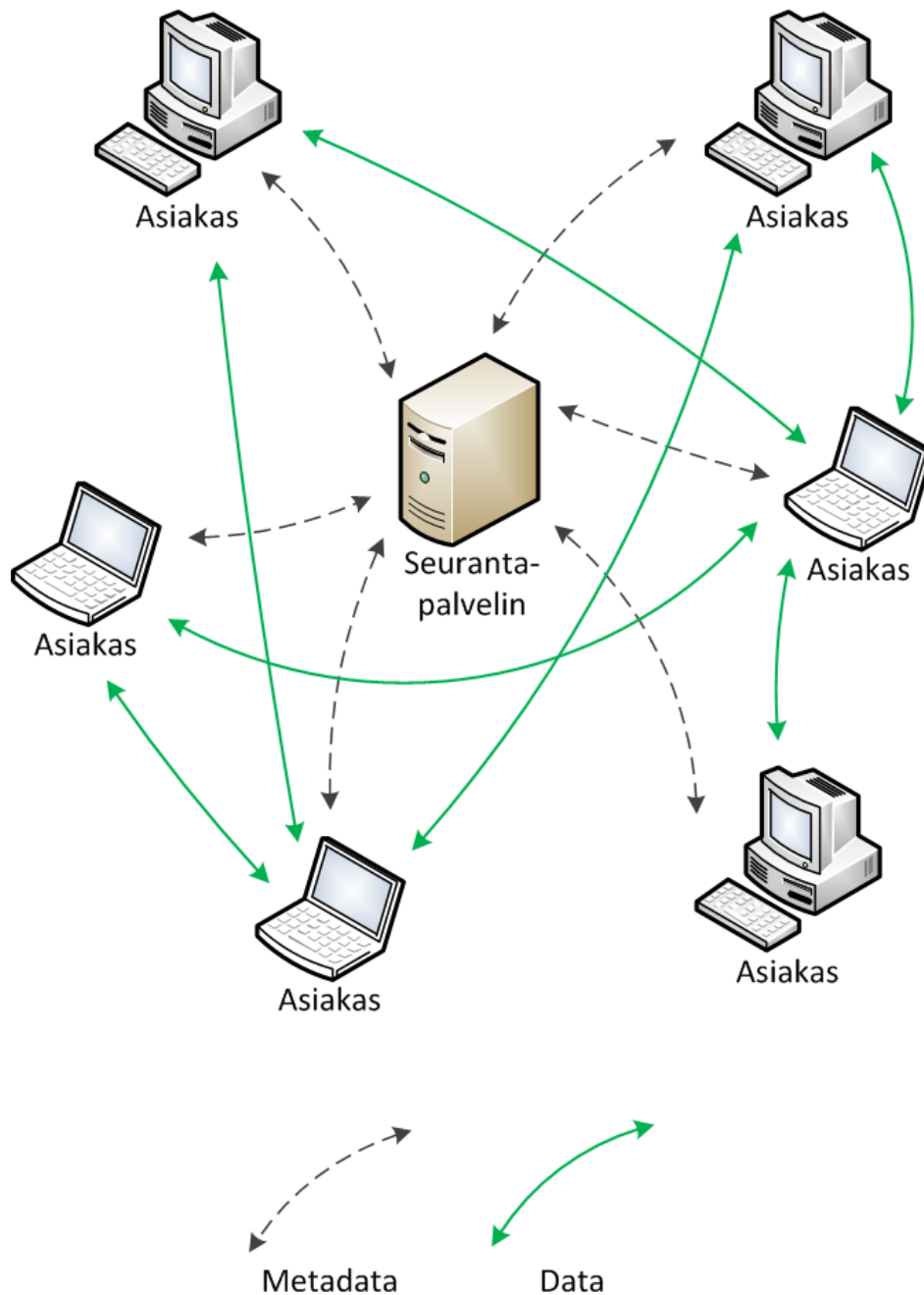
Seurantapalvelin (tracker) avustaa BitTorrent-protokollassa sisällönjakelua [Coh08]. Seurantapalvelimella ei ole itse sisältöä vaan sen tärkein tehtävä on jakaa asiakkaille tietoa siitä, keitä muita asiakkaita samasta sisällöstä on kiinnostunut.

Asiakasohjelma ottaa yhteyden seurantapalvelimeen käyttäen announce komentoa. Announce komennolla on kaksi päätehtävää:

- Asiakasohjelma ilmoittaa announcella haluavansa liittyä parveen ja paljonko sillä itsellään on valmiina haluttua sisältöä
- Seurantapalvelin vastaa announceen kertoen muiden samaa sisältöä jakavien asiakkaiden IP-osoitteet, porttinumerot ja paljonko niillä on valmiina haluttua sisältöä

Seurantapalvelimella täytyy olla jokin tapa tallentaa ja päivittää sisältökohtaisesti asiakkaiden tiedot, jotta se pystyy toimittamaan ne edelleen niitä haluaville asiakkaille. Lisäksi seurantapalvelimella on oltava tapa poistaa vanhentuneet asiakastiedot, jotta se ei toimita edelleen vanhentunutta tietoa. Yksi strategia vanhentuneen tiedon poistolle on poistaa esimerkiksi sellaiset sisältökohtaiset asiakastiedot, joissa asiakas ei ole tunnin sisällä ilmoittanut olevansa enää sisällöstä kiinnostunut.

Kuvassa 2 on esitetty perinteinen BitTorrent sisällönjako. Tietystä sisällöstä kiinnostuneet asiakkaat käyttävät BitTorrent-tiedostoa jakaakseen sisältöä. BitTorrent-tiedostossa olevaa seurantapalvelimen announce osoitetta käyttäen ne kommunikoivat seurantapalvelimen kanssa, joka puolestaan kertoo niille muiden asiakkaiden tiedot. Näitä tietoja hyväksikäyttäen asiakkaat välittävät haluttua sisältötietoa toisilleen.



Kuva 2. BitTorrent liikenteen esimerkkikuvaus

#### 2.7.4 Parvi

Parvi (swarm) on samasta BitTorrent-sisällöstä kiinnostuneiden asiakkaiden joukko. Parvi voi olla muodostunut käyttäen samoja seurantapalvelimia tai vaikkapa hajautetun tiivistetaulun kautta. Tärkeää ei kuitenkaan ole se miten parvi on muodostunut vaan se, että parven jäsenet tuntevat ainakin osajoukon toisistaan ja kykenevät näin jakamaan ja lataamaan sisältöä myös ilman ulkoista apua.

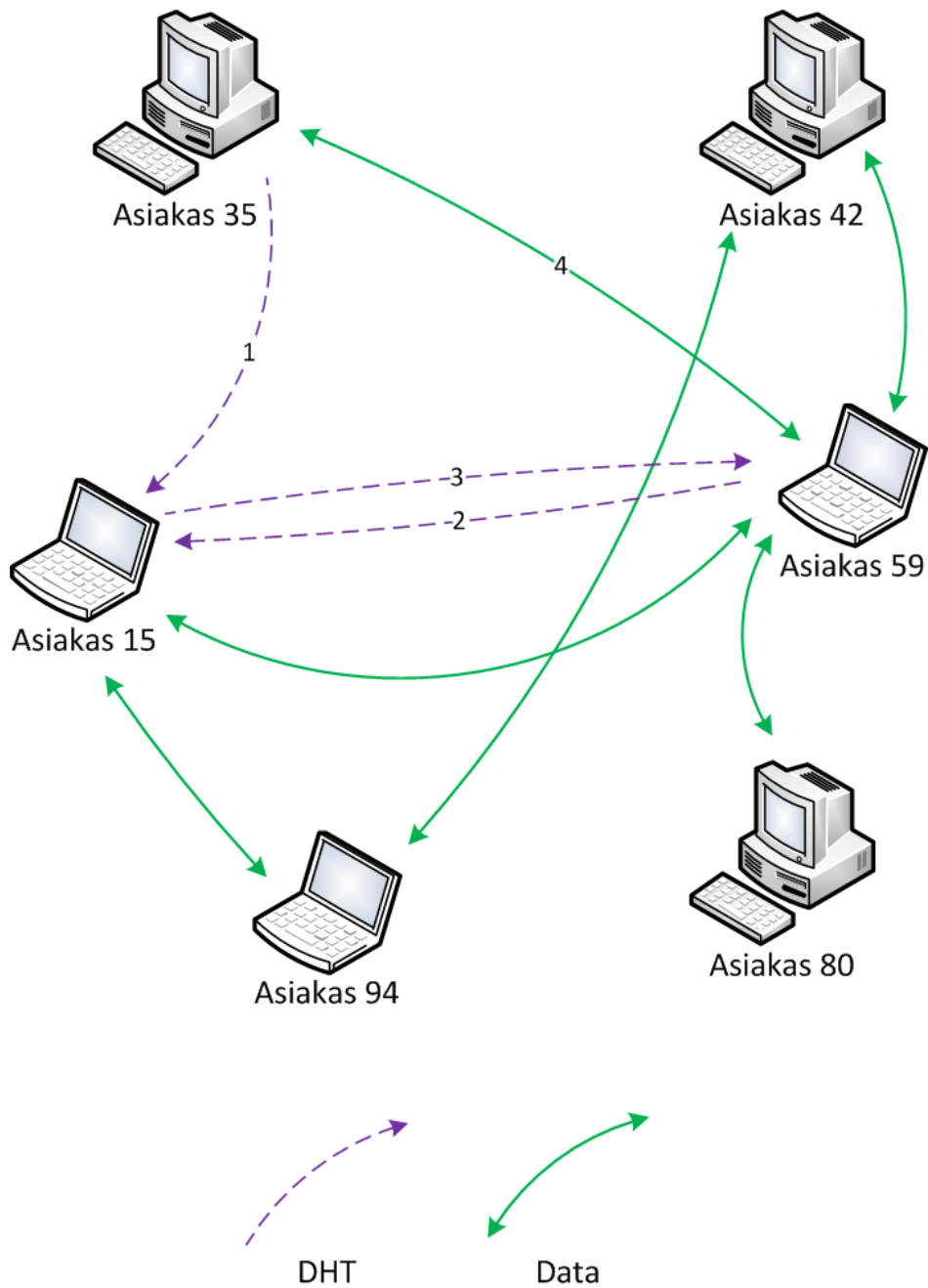
### 2.7.5 Hajautettu tiivistetaulu

Hajautettu tiivistetaulu (distributed hash table, DHT) on vertaisverkoissa käytetty tekniikka [LoN08]. Sitä käyttäen verkon solmut voivat itsenäisesti, ilman keskitettyä hallintaa, määrittää itselleen paikan verkossa ja löytää verkon muita solmuja ja sitä kautta haluamaansa sisältödataa.

Hajautettu tiivistetaulu -tekniikasta on monta eri versiota, mutta perusidea on kaikissa sama. Verkon jäsenet valitsevat itselleen (käytännössä) yksikäsitteisen ID:n, joka on samassa numeroavaruudessa sisältödatasta otettujen tiivisteiden kanssa. Tieto siitä, kellä on tietty sisältödata, ilmoitetaan verkon sellaisille solmuille, joiden ID on lähellä sisällytödatan tiivistettä. Kun joku muu verkon solmu tämän jälkeen haluaa ladata jotain tiettyä sisältödataa tarvitsee sen siis löytää verkosta sellaiset solmut, joiden ID on mahdollisimman lähellä datan tiivistettä ja kysyä niiltä datan sijaintia.

Kuvassa 3 on yksinkertaistettu esitys Hajautettu tiivistetaulu tekniikan toiminnasta BitTorrent-protokollalla.

1. Verkon solmulla ID 35 on sisältödataa, jonka tiiviste on 14. Lähin verkon solmun ID tuota lukua on solmu ID 15. Solmu ID 35 ilmoittaa solmulle ID 15 että sillä on sisältödataa, jonka tiiviste on 14.
2. Solmu ID 59 on kiinnostunut lataamaan sisältödataa, jonka tiiviste on 14. Se etsii verkosta solmu ID:n, joka on mahdollisimman lähellä tiivistettä 14. Selvitettään että solmu ID 15 on lähin, se kysyy siltä tietääkö se kenellä sisältödata on.
3. Solmu ID 15 vastaa solmu ID:lle 59, että sisältödata 14 on saatavissa solmu ID:ltä 35.
4. Solmu ID 59 pyytää solmu ID:ltä 35 sisältödataa tiivisteellä 14 käyttäen BitTorrent-protokollaa. Datasiirto käynnistyy. Jos sama sisältödata on usealla eri solmulla, ottaa solmu ID 59 yhteyden myös niihin.



Kuva 3. BitTorrent-liikenteen esimerkkikuvaus käytettäessä DHT:ta

## 2.8 Suoratoisto

Suoratoisto (streaming) tarkoittaa tiedon, yleensä äänen tai videon, lähettämistä datavirrana (stream) siten, että sitä voidaan toistaa kohteessa ennen kuin se on vastaanotettu kokonaan. Suoratoistostrategiat voidaan jakaa karkeasti kahteen kategoriaan; pyydettyyn (on-demand) tai reaaliaikaiseen.

Pyydettyssä strategiassa toistetaan olemassa olevaa tiedostoa (esimerkiksi videotalliota). Asiakas pyytää tiedostoa esimerkiksi menemällä Internet-sivulle, jolle on upotettuna

videotiedosto. Jos videovirtaa ei pystytä siirtämään niin nopeasti kuin sitä katsotaan, pysähtyy videon toisto hetkeksi. Toisto jatkuu, kun riittävästi uutta dataa on taas saatavilla.

Reaaliaikaisessa strategiassa toistetaan reaaliaikaista lähetystä (esimerkiksi urheilutapahtuma). Asiakas pyytää tiedostoa esimerkiksi menemällä Internet-sivulle, jolle on upotettuna videotiedosto. Jos videovirtaa ei pystytä siirtämään niin nopeasti kuin sitä katsotaan, hypätään mahdollisuuksien rajoissa toistamatta jääneen kohdan ohitse. Toistoa ei kannata pysäyttää, jotta reaaliaikaisuus saadaan säilytettyä. Lisäksi on mahdollista että toistamatta jäänyt data ei koskaan saavu ja sitä ei välttämättä pystytä saamaan enää uudelleen.

## **2.9 Selaimen lisäosa**

Internet-selaimet ovat hyvin monimutkaisia ohjelmistoja. Niiden täytyy tukea moninaista sisältöä ja tapaa siirtää dataa verkosta käyttäjälle. Nykyaikaisten Internet-selainten täytyy kuitenkin olla lisäksi laajennettavia, jotta esimerkiksi uudenlaisen sisällön tukeminen on mahdollista. Käyttäjien tavat käyttää Internet-selaimia ovat myös hyvin moninaisia ja Internet-selainten täytyy taipua käyttäjien toiveiden mukaisiksi. Selaimien lisäosat (plugin) tarjoavat yhden mahdollisuuden laajentaa selaimen käytettävyyttä tai sisällöntukea. Esimerkkinä selaimen lisäosasta toimivat vaikkapa mainosten esto-ohjelmat, jotka muokkaavat käyttäjälle näkyvää sisältöä. Lisäosat toimivat ikään kuin moduuleina alkuperäiselle selaimelle. Selaimen kehittäjän päätös on tukeeko selain vain esimerkiksi selaimen kehittäjän hyväksymiä lisäosia vai voiko kuka tahansa laajentaa selaimensa toimintaa omilla lisäosillaan.

Lisäosan asentaminen vaatii aktiivista panosta käyttäjältä. Vähintään lisäosan asentaminen selaimen vaatii asentamiskyselyn hyväksymistä, mutta asentaminen voi olla myös monimutkaisempi prosessi. Viruksia ja haittaohjelmia välttääkseen käyttäjä ei myöskään välttämättä halua asentaa lisäosaa koneelleen, vaikka sen olemassaolosta voisi olla hänelle hyötyäkin. Lisäosat ovat myös usein toteutettu vain jotain Internet-selaimen versiota vasten ja näin ollen niiden toimivuudesta selaimen muilla versioilla ei ole välttämättä takuuta. On näin ollen mahdollista, että aiemmin toiminut lisäosa lakkaa toimimasta, kun Internet-selain päivitetään uudempaan versioon.



## 2.10 HTML5

HTML5 on viides versio Hypertext Markup Language -kielestä [HBF14]. Se tuo monia uusia ominaisuuksia vanhoihin versioihin nähden kuten video- ja audioelementit. Lisäksi uudet sovelluskehikset, kuten WebRTC, pohjaavat HTML5 ominaisuuksiin. Käytännössä kaikki Internet-selaimet tukevat HTML5 kieltä, mutta eivät kaikkia sen lisäyksiä tai ominaisuuksia.

## 2.11 Media Source Extensions

HTML5 tukee median esittämistä joko paikallisesta lähteestä tai verkon yli palvelimelta. Tällöin kuitenkin lähteenä on edelleen joku tietty tiedosto/datavirta. Sitä luetaan alusta loppuun ja selainpäässä ei voida vaikuttaa sen sisältöön. Media Source Extensions (MSE) mahdollistaa mediavuon tuottamiseen selaimessa katsottavaksi Javascriptin avulla [WSW16]. Koska data ei tule enää tietyistä tiedostosta/datavirrasta, vaan ohjelmallisesti, voidaan monet perinteisen tavan rajoitteet rikkoa. Tärkeimpinä ominaisuuksina ovat mukautuvat videovuot ja aikasiirtymät.

### 2.11.1 Mukautuva videovuo

Koska videovuo kulkee Javascriptin kautta toistimeen, pystytään ohjelmallisesti havaitsemaan, jos dataa kulutetaan liian nopeasti siihen nähden miten sitä tuotetaan. Tai päinvastoin voidaan myös havaita se, jos dataa tulee sisään paljon nopeammin kuin sitä kulutetaan. Jos esimerkiksi samasta videotiedostosta on olemassa erilaatuisia versioita, voidaan Javascriptistä käsin siirtyä käyttämään eri videovuolähdettä. Javascriptissä voidaan siis mukautua automaattisesti ympäristön asettamiin rajoituksiin. Tässä tapauksessa mukautuma tehtäisiin siis asiakkaan verkon nopeuden mukaan.

### 2.11.2 Aikasiirtymät

Jos toistettava tiedosto on paikallinen tiedosto, pystyy videoistin mahdollisesti siirtymään sen sisällä ajallisesti eri kohtiin. Jos tiedosto ei ole paikallinen tai jos tiedoston sijaan toistetaan videovuota, ei tuo siirtymä enää onnistukaan. Javascript mahdollistaa sen, että tiedostoa tai videovuota aletaan lukemaan eri kohdasta ohjelmallisesti. MSE standardoi lisäksi sen tavan, millä tuohon eri kohtaan voidaan vuossa siirtyä.

### 2.11.3 Yleistä

Käytännön esimerkkinä MSE:n hyödyistä toimii vaikkapa videonjakopalvelu YouTube. Ennen vanhaan YouTube tarvitsi toimiakseen kolmannen osapuolen lisäosan; Adobe Flash Playerin. HTML5-standardia tukevien selainten myötä mahdollistui se, että YouTube pystyi tarjoamaan videoita myös suoraan, ilman lisäosia. Mutta vasta kun selaimet alkoivat tukea MSE:tä, on YouTube oikeasti pystynyt siirtymään HTML5:n käyttöön Flash Playerin sijaan. Kaikki selaimet eivät tue edelleenkaan MSE:tä ja niiden kohdalla tapahtuu taustalla piilossa ”fallback” Flash Playerin käyttöön. YouTube:n sivuilta löytyy testi, josta näkee kumpaa tapaa toistaa videoita oma selain käyttää [You16].

### 2.12 Java-sovelma

Java-sovelma (Java Applet) on Java ohjelmointikielellä kirjoitettu pieni sovellus. Java-sovelma on usein osa Internet-sivua, mutta sitä suoritetaan erillisessä prosessissa Java-virtuaalikoneen (JVM) avulla. Koska Java-virtuaalikone ei ole osa selainta, tarkoittaa se käytännössä sitä, että JVM on pitänyt asentaa erikseen selaimen lisäosaksi. Vaikka Java-koodi itsessään on alustariippumatonta vaativat Java-sovelmat toimiakseen siis erikseen asennettavan lisäosan asentamisen. Javan kehittäjä, Oracle, on ilmoittanut että tulevassa JVM versiossa 9 lakkautetaan tuki Java-sovelma lisäosalle [Kri16].

### 2.13 Java Web Start

Java Web Start (JWS) on samantyylinen kuin Java-sovelma siinä, että se käynnistetään Internet-sivulta selaimen kautta. Se eroaa kuitenkin Java-sovelmasta siinä, että sitä ei suoriteta osana Internet-sivua vaan täysin omana sovelluksenaan. Se on ikään kuin työpöytäsovellus, joka vain ladattiin verkosta suorituksen ajaksi. Vaikka JWS-sovelluksessa onkin rajoitteita siinä mihin kaikkiin resursseihin se pääsee käsiksi, tulevat nuo rajoitteet Javasta käsin, eivätkä selaimesta. JWS-sovelluksia ei suoriteta selainten lisäosilla vaan erikseen asennetulla Java-virtuaalikoneella. Java-virtuaalikoneen esi-asennus on siis pakollista JWS-sovelluksia suorittamista varten. JWS-sovellukset toimivat kuitenkin käytännössä kaikilla nettiselaimilla ja kaikissa sellaisissa käyttöjärjestelmissä, joihin on olemassa Java-virtuaalikonetoteutus.

## **2.14 NAT**

Kun kaksi IP-verkossa olevaa laitetta kommunikoi keskenään, ne lähettävät toisilleen paketteja. Kussakin paketissa on lähettäjän ja vastaanottajan IP-osoitteet ja yhteyden ajan käytössä oleva porttinumero. Jos laitteet ovat keskenään eri verkoissa, voi verkosta toiseen siirryttäessä tapahtua osoitteenmuunnos. Kun paketti saapuu verkkojen rajalle, siinä olevan reitittimen pääasiallinen tarkoitus on välittää paketti eteenpäin toiseen verkkoon siten, että se löytää kohdekoneelle. Jos lähtöverkossa on käytössä osoitteenmuunnos, kirjoittaa reititin (NAT (Network Address Translator)) pakettiin lähtöosoitteeksi uuden osoitteen, jonka se tallentaa sisäisesti reititystauluihinsa. Jatkossa kaikkiin saman lähtö- ja kohdeosoitinparin paketteihin kirjoitetaan lähtöosoite reititystaulun mukaisesti. Jos taas kääntäen reitittimeen saapuu paketti, jonka kohdeosoite löytyy sen reititystauluista ja vaatii osoitteenmuunnosta, kirjoittaa reititin kohdeosoitteen yli ja lähettää sen eteenpäin kohdeosoitteen määräämään paikkaan. Yksityishenkilöiden kotiverkkojen ollessa kyseessä osoitteenmuunnos tapahtuu tyypillisimmin, kun kotiverkon tietokoneelta saapuva paketti saapuu ADSL-modeemiin tai muuhun paketin yleiseen internetiin reitittävään laitteeseen.

## **2.15 ICE**

On hyvin yleistä, että satunnainen asiakaskone on NATin takana. Jos kaksi asiakaspään konetta haluaa muodostaa yhteyden toisiinsa (esimerkiksi p2p), voi toinen tai jopa molemmat koneista olla NATin takana. Koska NAT-laitteessa osoitteenmuunnoksen tiedot kirjoitetaan reititystauluihin aina lähettäjän osoitetta muuttaen, voi ulkopuolelta tulevan yhteydenoton luominen olla vaikeaa tai jopa mahdotonta. ICE (Interactive Connection Establishment) tarjoaa standardoidut keinot ratkaista NATin läpi kulkemisen aiheuttamat ongelmat [Ros10].

### **2.15.1 STUN**

STUN (Session Traversal Utilities for NAT) metodien päämääränä on avustaa yhteyden muodostuksessa NATin takana olevien asiakkaiden välille [RMM08]. STUNia käyttäen asiakas saa tietää miltä sen IP-osoite ja portti näyttävät ulospäin NAT-muunnoksen jälkeen. Kun NATin takana oleva asiakas haluaa ottaa yhteyden toiseen asiakkaaseen, se lähettää ensin kyselyn STUN-palvelimelle. STUN-palvelin vastaa asiakkaalle kertoen miltä sen IP-osoite ja portti näyttävät sen näkökulmasta. Sen jälkeen asiakas voi välittää

eteenpäin toiselle asiakkaalle nämä tiedot yhteyden muodostusta varten. Jos vastapää on myös NATin takana, tekee hän samat toimenpiteet.

STUN ei kuitenkaan toimi kaikkien NAT tyyppien läpi eikä tapauksissa, joissa molemmat asiakkaat ovat saman NATin takana. Tämä rajoittaa sen käytettävyyttä.

## **2.15.2TURN**

TURN (Traversal Using Relay NAT) on ikäänkuin laajennos STUN-metodeille. Päämäärä on sama kuin STUNissa eli miten saada yhteys muodostettua NATin takana olevien asiakkaiden välille. Asiakkaat ottavat yhteyden toistensa sijaan TURN-palvelimeen. Kaikki liikenne asiakkaiden välillä kierrätetään TURN-palvelimen kautta. TURN toimii sellaisissakin tilanteissa joissa STUN ei toimi, joten se on luotettavampi ratkaisu. Haittapuolena TURNissa on kuitenkin runsas, ja eräissä mielessä turha, palvelinresurssien käyttö. TURN ratkaisu onkin vasta toissijainen tapa toimia ja ensin kannattaa koittaa yhteyden luomista STUNin avulla.

## **3 Mahdolliset lähestymistavat**

Tässä luvussa pohditaan erilaisia ratkaisuvaihtoehtoja esitettyyn ongelmaan. Erilaiset ratkaisut on jaoteltu kukin omaan lukuunsa. Jokaisessa luvussa esitetään jokin tapa ratkaista ongelma ja sen mahdolliset muunnelmät. Jotta ratkaisuja voidaan verrata keskenään, tarvitaan myös kriteeristö sille, mitä ratkaisulta toivotaan ja kääntäen mitä asioita ratkaisussa pitäisi välttää.

Kaikki esitellyt ratkaisuvaihtoehdot pohjaavat jossain määrin mukautuvan sisällönjakelun ja vertaisavusteisen sisällönjakelun periaatteisiin. Kaikissa vaihtoehdoissa ns. bootstrap, eli toiminnan aloitus, vaatii videon siirtoa palvelimelta käyttäjälle muodossa tai toisessa. Ja toisaalta kaikissa vaihtoehdoissa videon pääasiallinen siirtotapa on käyttäjiltä toisille käyttäjille [CSD07].

Aluksi käydään läpi käytettävä kriteeristö. Sen jälkeen käydään läpi erilaiset ratkaisuvaihtoehdot ja miten kukin niistä suhtautuu kriteeristöön nähden. Lopuksi vertaillaan ratkaisuja keskenään ja tehdään valinta niiden välillä.

### **3.1 Kriteeristö**

Kriteeristö voidaan jaotella kahteen pääkategoriaan sen mukaan, katsotaanko asiaa pal-

velun tarjoajan vai palvelun käyttäjän näkökulmasta.

Palveluntarjoaja:

- Resurssien käytön määrä
  - Kaistankäyttö
  - Prosessoritehon- ja muistinkäyttö
  - Palvelimien määrä
- Toteutuksen monimutkaisuus ja ylläpidettävyys

Palvelun käyttäjä:

- Palvelun käyttöönoton helppous
- Palvelun käyttämisen helppous
- Palvelun toimivuus
- Palvelun nopeus
  - Objektiivinen
  - Subjektiviinen
- Palvelun saatavuus
- Toimivuus mobiilipuolella

On tärkeää että ratkaisu ottaa kummankin osapuolen näkökulmat huomioon. Toisaalta koko gradussa käsiteltävän ongelman voi ajatella olevan palvelun tarjoajan ongelma, koska ratkaisussa pyritään löytämään halpa, mutta toimiva tapa toimittaa videoita käyttäjälle [ZZF12]. Toisaalta kuitenkin ongelma on myös käyttäjän ongelma, koska esimerkiksi palvelun toimivuus ja käytettävyys näkyvät nimenomaan käyttäjälle. Huonosti toimiva ratkaisu myös karkottaa käyttäjiä ja se ei myöskään ole palvelun tarjoajan intresseissä.

### **3.2 HTML5 sovellettuna**

Ennen monimutkaisempiin ratkaisuihin siirtymistä on hyvä tarkistaa, voiko kuvatus ongelman ratkaista jo olemassa olevassa ympäristössä. Koska HTML5 on jo levinnyt laajalle, ja sitä tuetaan monissa selaimissa, niin on kenties hyväksyttävää rajoittua tut-

kimaan sen tarjoamia mahdollisuuksia. Toinen mahdollisuus olisi tutkia selainkohtaisesti mitä HTML5 ominaisuuksia se tukee, jotta saataisiin aikaiseksi ratkaisu, joka toimii varmasti kaikissa selaimissa. Koska HTML5 tuki vain kasvaa ajan myötä hyväksytään tässä tarkastelussa lähtökohdaksi täysi HTML5 tuki.

Videonkatselumekanismit ovat HTML5:ssä sisäänrakennettuna. On siis selvää että videoita itsessään voidaan katsoa puhtaalla HTML5 toteutuksella. Itseasiassa kaikki muut ehdotetut ratkaisut käyttävät HTML5:sen videoistinta.

HTML5 ei pysty rikkomaan yhtä websovellusten perusrajoitteista eli saman alkuperän käytäntöä. Tämä tarkoittaa tavoitellun ratkaisun tapauksessa sitä, että video pitää ladata yksittäisestä osoitteesta, koska ohjelmallisesti ei voida yhdistää useasta eri lähteestä saapuvaa videotiedostoa. Käytännössä siis lähteitä videotiedostolle voi olla vain yksi kappale ja useamman lähteen käyttö on poissuljettu mahdollisuus.

Toinen ongelma puhtaassa HTML5 ratkaisussa on se että HTTP-liikenteen aloittaja on aina asiakaspää. Asiakas pyytää resurssia ja saa sen palvelimelta vastauksena. Asiakas ei itsessään pysty kuitenkaan toimimaan palvelimen roolissa eli asiakas ei itse pysty tarjoamaan ulospäin resurssia muille käyttäjille. Tämä ei sinänsä ole puhdas HTML5 asia vaan yleinen rajoite HTTP-protokollassa.

Yllä mainituista syistä johtuen puhdas HTML5 toteutus ei siis ole mahdollista. Ratkaisun kriteerien pohtiminen on periaatteessa turhaa, mutta ainakin yksi kohta voidaan nostaa esiin. Jos toteutus olisi tehty puhtaasti jo olemassa olevalla ympäristöllä, olisi se käyttöönoton suhteen kaikkein paras ratkaisu asiakkaan kannalta. Asiakkaan ei tarvitsisi tehdä mitään toimenpiteitä ja ratkaisu toimisi missä tahansa ympäristössä ja selaimessa.

### **3.3 Selaimen lisäosa**

Selaimen lisäosa toteutuksessa voitaisiin edetä esimerkiksi siten, että selaimen lisäosa olisi täysin toimiva BitTorrent-asiakasohjelmisto, joka osaisi myös tuottaa videovuon halutusta sisällöstä ja näyttää sen selaimessa videoistimessa. Selaimen lisäosana toimivia BitTorrent-asiakasohjelmistoja on olemassa, joten ratkaisu on siltä osin jo toteutettu [ZCB11] [HSR07]. Ei ole kuitenkaan olemassa mitään selainriippumatonta ratkaisua, joka toimisi kaikissa tilanteissa. Video oltaisiin voitu ladata yleiseen verkkoon muutamalle BitTorrent-asiakasohjelmalle (jotka toimisivat palvelimilta käsin) etukäteen, jotta videon lataus voitaisiin aloittaa suoraan noilta asiakkailta BitTorrent-

protokollalla [KSK08]. Vaihtoehtoisesti ensimmäiset sivulle tulevat asiakkaat voisivat toimia prosessin aloittajina (ns. bootstrap) eli ne voisivat ladata videon ensin normaalisti HTTP TCP-siirrolla ja sen jälkeen toimia ensimmäisinä jakajina käyttäen BitTorrent lisäosaa.

Käyttäjiä on vaikea saada käyttämään selaimen lisäosaa. Esimerkiksi vuonna 2010 Firefox-selaimen suosituimman lisäosan, Adblock Plussan, oli asentanut vain 4,2% kaikista Firefox-selaimen käyttäjistä [ZZF12].

Jos sivustoa ei haluta rajoittamaan toimimaan vain yhden Internet-selaimen yhdellä versiolla, tarkoittaa lisäosaan turvautuminen sitä, että sen pitää tukea useaa eri selainta ja vieläpä useaa eri selainversiota. Tämän toteuttaminen ei välttämättä ole halpaa tai tietoturvallista. Oman lisäosan tietoturvan ylläpitäminen vaatii alan trendien ja uusien uhkien seuraamista ja tähän ei välttämättä ole resursseja. Yleisemmissä lisäosaratkaisuihin vastuu voi olla jaettu tai se voi olla jopa kokonaan toisella taholla.

Koska selaimen lisäosassa eivät päde samat rajoitteet kuin normaalissa verkkosivustossa, pystytään lisäosaa käyttäen paljon enempään kuin ilman sitä. Jotta sivusto toimisi kuitenkin täydellisesti, pitäisi lisäosa olla toteutettuna monelle eri käyttöjärjestelmälle ja selaimelle. Tämä ei ole mahdollista ilman suurta rahallista ja ajallista panostusta. Pienemmällä vaivalla todennäköisemmin päästäisiin, jos ratkaisu olisi alun perin toteutettu perinteisemmin kaistaa ja palvelimia lisäämällä.

Palvelun käyttäjälle selaimen lisäosa vaatii lisäosan asentamisen selaimen palvelua käyttöönotettaessa. Kuten jo yllä todettiin, niin se jo itsessään on ongelmallista. Kun lisäosa on otettu käyttöön, on palvelun käyttö todennäköisesti kuitenkin helppoa. Jos lisäosa on toteutettu hyvin, ja selain itsessään ei laita lisäosan toimintaan liikaa rajoituksia, voi lisäosa toimia taustalla täysin huomaamattomasti ja käyttäjän näkökulmasta kaikki ”vain toimii”.

Selaimien lisäosia ei yleensä ole mahdollista toteuttaa mobiilipuolen selaimiin. Tämä asia voi muuttua tulevaisuudessa, mutta noin yleensä ottaen mobiiliselaimet eivät tue mitään lisäosia.

### **3.4 Java-sovelma**

Tämä ratkaisu lähtisi täsmälleen samasta lähtökohdasta kuin selaimen lisäosa. Jopa siinä määrin, että ratkaisu, ja siihen liittyvät ongelmat, ovat hyvin pitkälti samanlaisia. Java-

sovelma vaatii toimiakseen etukäteen asennetun Java-lisäosan ja lisäksi itse sovelma pitää ladata palvelimelta ennen käyttöönottoa joka kerta uudelleen. Usein selaimet myös kysyvät käyttäjältä joka suorituskerralla lupaa suorittaa sovelma. Java-sovelman etuna on se, että se on hyvin standardoitu tapa tehdä lisää toiminnallisuutta selaimeen ja siksi voidaan periaatteessa lähteä siitä, että ratkaisu on selainriippumaton. Valmiita Java-sovelmina toteutettuja BitTorrent-asiakasohjelmistoja on olemassa, joten ratkaisu on siltä osin jo toteutettu. Java-sovelmien suorituksesta löytyy kuitenkin jatkuvasti tietoturvaheikkouksia ja siksi niiden käyttöä ei suositella. Lisäksi Javan sovelluskielen kehittäjä on julkisesti ilmoittanut lopettavansa Java-sovelmien tuen jatkossa.

Palvelun tarjoajan näkökulmasta Java-sovelma on helpompi toteuttaa ja ylläpitää kuin jokaiseen eri käyttöjärjestelmään ja Internet-selaimeen tehty erillinen lisäosa. Jatkuvasti löytyvien uusien tietoturvaheikkouksien takia voi ylläpito kuitenkin osoittautua aikaa vieväksi. Koska toteutus pitää joka käyttökerralla ladata kokonaisuudessaan palvelimelta selaimelle ei yksittäisten videoiden tapauksessa välttämättä säästetä kunnolla edes palvelinresursseja. Hyötyä syntyy kunnolla vasta jos käyttäjä katsoo peräkkäin useampia eri videoita.

Palvelun käyttäjän näkökulmasta selaimen jatkuvasti pyytämä lupa suorittaa Java-sovelma on varmasti häiritsevää. Todennäköisesti mobiililaitteissa ei tätä käyttötapaa saada toimimaan ollenkaan, joten ratkaisu on rajoitettu tietokonekäyttöön.

### **3.5 *Java Web Start***

Tämä ratkaisu vaatisi toimiakseen sen, että käyttöjärjestelmään on asennettu Java-virtuaalikone. Samoin kuin Java-sovelmissa, pitää ohjelma ensin ladata koneelle palvelimelta, sen käyttö hyväksyen, mutta jatkossa ohjelma säilyy koneella ja sitä ei tarvitse erikseen ladata. Huonona puolena applettiin nähden on se, että ilmeisesti haettua videota on vaikea streamata sivuston itsensä kautta, koska sovellus toimii ihan erillisenä ohjelmana. Videon voi toki näyttää ohjelmassa itsessään, mutta silloin videon näyttämisenkin pitäisi toteuttaa siinä valmiiden HTML5 komponenttien sijaan.

Palvelun tarjoajan näkökulmasta resurssien käytön määrä vähenee, jos käyttäjä katsoo useampia videoita tai jos yksittäisten videoiden koko on suuri. Koska koko ohjelma pitää ladata palvelimelta käyttäjän koneelle, niin alkukustannus on suuri. Samoin kuin Java-sovelmissa pitäisi toteutuksen ja ylläpidettävyyden olla esimerkiksi selaimen lisä-



osaan nähden helpompaa. Tämä johtuu siitä, että Java-virtuaalikoneen toteutus on jo olemassa eri käyttöjärjestelmille ja sen toiminnan pitäisi olla standardoitua. Jonkin verran variaatiota varmasti kuitenkin on eri ympäristöjen ja versioiden välillä. Koska sovellusta ei ladata joka käyttökerralla palvelimelta asti pitää tämä ottaa huomioon jos sovelluksesta tulee uusia versioita.

Palvelun käyttäjän näkökulmasta tämä ratkaisu on ehkä kaikkein monimutkaisin. Ei riitä että selaimen asennetaan jotain uutta toiminnallisuutta vaan Java-virtuaalikone asennetaan kokonaan erikseen käyttöjärjestelmätasolla. Lisäksi sovellusta suoritetaan kokonaan omassa ikkunassaan ja se varmasti hämmentää peruskäyttäjää. Todennäköisesti mobiililaitteissa ei tätä käyttötapaa saada toimimaan ollenkaan, joten ratkaisu on rajoitettu tietokonekäyttöön.

### **3.6 WebRTC**

WebRTC:n käyttöä nettisivuston staattisten osien jakamiseen on jo tutkittu [ZZM13]. Toteutuksessa voitaisiin lähteä esimerkiksi siitä, että video olisi alun perin ladattavissa palvelimelta normaaliin tyyliin, ja sen jälkeen sitä jakaisivat eteenpäin ne käyttäjät, jotka videon ovat jo ladanneet. Jollakin tavalla täytyy pitää tiedossa ketkä kaikki käyttäjät videota ovat lataamassa ja keillä se on valmiina. Tämän tiedon pohjalta voidaan puolestaan päättää kuka käyttäjä jakaa tiedoston kenellekin. Periaatteessa luontevin paikka tallentaa tämä tieto ja tehdä päätöksiä olisi palvelin itse. Se pystyy keskitetysti näkemään kokonaiskuvan kaikkein helpoimmin ja tekemään oikeudenmukaisimmat ja tehokkaimmat päätökset. Tällöin kuitenkin palvelimen käyttö lisääntyy ja siitä juuri ratkaisussa pyrittiin eroon. Jos taas tieto tallennetaan hajautetusti kaikille käyttäjille, niin käyttäjien tasapuolinen kohtelu vaikeutuu. Voi käydä esimerkiksi niin, että osalta käyttäjistä tiedosto ladataan useampaan kertaan samalla kun joiltakin käyttäjiltä ei kertaakaan.

Jos käyttäjän selain ei tue vaadittuja WebRTC tai MSE ominaisuuksia voidaan tehdä fallback ja käyttäjälle näytetään video suoraan palvelimelta. Tämä on mahdollista tehdä käyttäjälle näkymättömästi. Luonnollisesti tällöin videon lataamisen nopeus riippuu palvelimen resursseista.

Jos käyttäjien yhtäaikainen määrä on suuri, voidaan suurin osa videon siirrosta siirtää asiakkaiden vastuulle. Tämä tarkoittaa selkeää vähennystä kaistankäytössä ja kaikissa muissakin resursseissa palvelinpäässä.

Toteutuksen monimutkaisuutta lisää se, että päätöksentekologiikka (siitä kuka käyttäjä jakaa tiedostoja kellekin) pitää suunnitella ja toteuttaa itse. Samoin käyttäjien yhtäkkisten poistumisten sivustolta ja muiden vastaavien erikoistilanteiden käsittelyyn pitää varautua omassa toteutuksessa.

Käyttöönotto ja käyttö on helppoa sillä asiakkaan näkökulmasta palvelu toimii täsmälleen samoin kuin jos hän käyttäisi tavallista videonjakopalvelua. Palvelun toimivuus ja saatavuus puolestaan riippuvat hyvin pitkälti palvelun käyttäjämääristä ja siitä kuinka pitkäksi aikaa ihmiset jäävät sivustolle videon katsottuaan. Koska videota ladataan toiselta käyttäjältä, voi yksittäisen käyttäjän saama palvelu hidastua todella huomattavasti jos jakaja yhtäkkiä poistuu paikalta. Samoin ollaan riippuvaisia vahvasti yksittäisen toisen käyttäjän upload-kaistan määrästä. Siksi eri käyttäjien näkemykset palvelun laadusta voivat vaihdella huomattavastikin. Samoin saman käyttäjän peräkkäisten videoiden latausnopeuksissa voi olla suuret erot.

### **3.7 *WebRTC ja BitTorrent***

WebRTC ja BitTorrent yhdistelmätoteutuksessa voitaisiin edetä hyvin samaan tapaan kuin pelkkää WebRTC:tä käyttävässä toteutuksessa. Video olisi ensin ladattavissa palvelimelta, ja sen jälkeen kun muutama käyttäjä on sen itselleen ladannut, voisivat muut käyttäjät ladata sitä suoraan heiltä. Erona puhtaaseen WebRTC toteutukseen olisi kuitenkin se että tiedostoa jaettaisiin käyttäjien kesken BitTorrent-protokollalla [SRT08]. Uusi käyttäjä pystyy tällöin lataamaan tiedostoa useammalta eri käyttäjältä yhtä aikaa ja pienemmissä palasissa. Vanhojen käyttäjien poistuminen sivustolta ei enää vaikuta latausnopeuksiin tai videonjaon yleiseen toimivuuteen niin paljoa, koska yksittäiset käyttäjät eivät ole enää niin keskeisessä asemassa.

BitTorrent-protokollaa ei voi käyttää suoraan Internet-selaimesta käsin, koska normaalisti käyttäjiltä käyttäjille suuntautuva kommunikaatio on mahdotonta. WebRTC on kuitenkin tarkoitettu nimenomaan käyttäjien väliseen kommunikaatioon, joten BitTorrent-protokollaa voidaan muokata käyttämään WebRTC:tä. WebRTC käyttää tiedonsiirrossa websocket-protokollaa ja BitTorrent puolestaan käyttää TCP- ja UDP-protokollia. Tämä tarkoittaa käytännössä sitä, että muokatut WebRTC-BitTorrent-asiakkaat eivät pysty kommunikoimaan normaalien BitTorrent-asiakasohjelmien kanssa, vaan ovat rajoittuneita yhteyksissään muihin WebRTC-BitTorrent-asiakkaisiin.

Jos käyttäjien yhtäaikainen määrä on suuri, voidaan suurin osa videon siirrosta siirtää asiakkaiden vastuulle. Tämä tarkoittaa selkeää vähennystä kaistankäytössä ja kaikissa muissakin resursseissa.

Toteutuksen monimutkaisuutta lisää se, että BitTorrent-protokolla vaatii toimiakseen joko seurantal palvelimen tai DHT:n käytön. BitTorrent-protokolla lisää myös liikenteen määrää sekä käyttäjien välillä että käyttäjien ja mahdollisen seurantal palvelimien välillä. WebRTC itsessään tuo signaloinnin tarpeellaan yhden kerroksen lisää metadatan siirron tarvetta, joten BitTorrent-protokollan kanssa metadatan siirtomäärät ovat normaalia suuremmat.

Käyttöönotto ja käyttö on helppoa sillä asiakkaan näkökulmasta palvelu toimii täsmälleen samoin kuin jos hän käyttäisi tavallista videonjakopalvelua. Palvelun toimivuus ja saatavuus puolestaan riippuvat pitkälti palvelun käyttäjämäärästä, mutta eivät yhtä paljon kuin puhtaassa WebRTC toteutuksessa. BitTorrent-protokolla mahdollistaa usealta eri käyttäjältä lataamisen samanaikaisesti ja yksittäisen käyttäjän poistuminen parvesta ei aiheuta ongelmia muille käyttäjille. Koska eri videoiden parven koko vaihtelee, voi saman käyttäjän peräkkäisten videoiden latausnopeuksissa olla suuret erot. Jos kuitenkin rajoitetaan tutkimaan viraaleja videoita, joissa parven koko on oletuksena suuri, pitäisi palvelun nopeus ja saatavuus olla suhteellisen vakio.

### **3.8 Vertailua ja valinta**

Tässä luvussa vertaillaan edellä käytyjä vaihtoehtoja ja valitaan toteutettava ratkaisu.

#### **3.8.1 Vertailun läpikäynti**

Taulukossa 1 on eri lähestymistapoja vertailtu keskenään kriteeristöä käyttäen. Vaihtoehdot kullekin kohdalle ovat arvot 0, 1 ja 2 siten, että 0 edustaa huonointa vaihtoehtoa ja 2 edustaa parasta vaihtoehtoa. Joissakin kohdissa arvotus on helppoa ja numeroarvon antaminen asialle on ollut suoraviivaista. Esimerkkinä tällaisesta käy vaikkapa käyttöönoton helppous millä tahansa vaihtoehdolla. Joissakin kohdissa arvotus on puolestaan hyvinkin vaikeaa ja on jouduttu tekemään edistyneitä arvauksia. Esimerkkeinä tällaisista käyvät vaikkapa toimivuuden ja nopeuden arviointi.

	HTML5	Selaimen lisäosa	Java- sovelma	JWS	WebRTC	WebRTC ja BitTorrent
Kaistankäyttö	2	2	1	2	2	1
Prosessori ja muisti	2	1	1	1	2	2
Palvelimien määrä	2	2	1	2	2	2
Monimutkaisuus ja ylläpidettävyys	2	0	0	0	1	2
<b>Yhteensä palveluntarjoaja</b>	<b>8</b>	<b>5</b>	<b>3</b>	<b>5</b>	<b>7</b>	<b>7</b>
Käyttöänoton help- pous	2	0	0	0	2	2
Käyttämisen helppous	2	2	0	1	2	2
Toimivuus	2	1	1	1	1	2
Nopeus	1	1	0	1	1	1
Saatavuus	1	2	2	2	0	2
Mobiilitoimivuus	2	0	0	0	1	1
<b>Yhteensä palvelun käyttäjä</b>	<b>10</b>	<b>6</b>	<b>3</b>	<b>5</b>	<b>7</b>	<b>10</b>
<b>50% / 50% painotettu keskiarvo</b>	<b>1,8</b>	<b>1,1</b>	<b>0,6</b>	<b>1,0</b>	<b>1,5</b>	<b>1,7</b>

Taulukko 1. Lähestymistapojen vertailua asteikolla nollasta kahteen

Taulukossa on eritelty toisistaan palveluntarjoajan näkökulma ja palvelun käyttäjän näkökulma. Loppuarvosanaa antaessa kummankin näkökulman pisteet on yhdistetty siten, että kummallekin on annettu sama painoarvo. Perusteluna tälle on se, että palveluntarjoajan ja palvelun käyttäjän tarpeet nivoutuvat ratkaisussa läheisesti yhteen. Ilman palvelua ei ole käyttäjiä ja toisaalta huonosti toimiva palvelu karkottaa käyttäjät. Palvelu, jolla ei ole käyttäjiä, on turha.

HTML5 ratkaisu eroaa kaikista muista ratkaisuista siinä, että sitä ei pysty toteuttamaan nykyisellään. Se on kuitenkin otettu mukaan taulukkoon tuottamaan lisää näkökulmia vertailuun. Koska toteutus on mahdotonta tehdä HTML5:sta käyttäen, ovat numeroarvot kenties kaikkein eniten arvattuja. Mikään ei kuitenkaan takaa sitä, että tämä ratkaisu olisi ilman muuta paras ratkaisu, joten kaikkia kriteerejä ei ole merkattu summassa parhaimmalla lukuarvolla. Esimerkiksi saatavuus voi jollakin paremmalla toteutusidealla olla huomattavastikin parempi kuin pelkällä naiivilla suoralla ratkaisulla. Toisaalta kuitenkin käyttämisen helppous on todennäköisesti paras, koska selaimet tukevat järjestelmää valmiiksi ilman muutoksia.

Tällä hetkellä mikään esitellyistä ratkaisuksista ei toimi mobiilipuolella. Tämä on todella suuri rajoite sillä yhä useammin videoita katsellaan myös mobiililaitteilla. Video muuttuu viraaliksi yleensä sosiaalisessa mediassa jaettujen linkkien avulla ja sosiaalista mediaa käytetään sitäkin yhä useammin puhtaasti mobiililaitteilla. Vaikka jokin ratkaisu toimisikin mobiililaitteilla, ei se välttämättä toimisi riittävän nopeasti ollakseen käyttökelpoinen ratkaisu. Matkapuhelinverkoissa olevien mobiililaitteiden ulospäin liikkuvan liikenteen kaista on usein todella rajoitettua, sillä normaalisovelluksissa vain sisäänpäin liikkuvan liikenteen kaistan nopeudella on väliä. Kaikki ratkaisut lähtevät siitä, että käyttäjät jakavat kohteena olevaa videosisältöä toisilleen, joten verkko, jonka solmut koostuvat suurimmaksi osaksi mobiililaitteista, on huomattavasti hitaampi kuin tietokoneista koostuva verkko.

Vaikka mikään ratkaisusta ei toimi mobiilipuolella, voidaan olettaa, että jossain vaiheessa jokin ratkaisu kuitenkin tulee mobiilipuolella toimimaan. HTML5 ratkaisussa voidaan tämä oletus katsoa jopa hyvin vahvaksi, sillä täysi HTML5 tuki mobiililaitteen selaimessa tuottaisi tämän automaattisesti. WebRTC toteutuksiin perustuvat ratkaisut ovat samoin vahvoilla, koska WebRTC on standardoitu ja standardin toteuttavia selaimia tulee ajan kanssa lisää, kenties jossain vaiheessa myös mobiilipuolelle. On myös mahdollista tehdä jokaiselle mobiililustalle oma sovellus, johon siirrytään automaattisesti, kun saavutaan jollekin palvelun videosivulle. Näin toimivat itseasiassa usein myös asiakas-palvelin –mallia noudattavat videonjakosivustot (esimerkiksi Youtube). Tätä mahdollisuutta ei kuitenkaan tutkita tässä gradussa tarkemmin ja se ei joka tapauksessa liity suoraan siihen mikä lähestymistapa valitaan. Mobiilisovellus on oma ohjelmansa ja sitä eivät sido Internet-selainten rajoitteet. Sosiaalisessa mediassa tapahtuvaa sisällönjakoa, mobiilitoimivuuden kannalta ja muuten, on myös tutkittu laajemminkin [EDB08].

### **3.8.2 Valinta**

Palveluntarjoajan näkökulmasta parhaimmiksi erottuvat HTML5 ja WebRTC pohjaiset ratkaisut. Tämä selittyy suurimmaksi osaksi sillä, että näissä ratkaisuissa monimutkaisuus ja ylläpidettävyyys ovat parhaimmalla tasolla. Toteutus on tehty kerran palvelimelle ja eri selaimille ei tarvita eri versioita. Huonoimmaksi ratkaisuksi puolestaan valikoitui Java-sovelma. Jatkuva tarve ylläpidolle ja jatkuva tarve ladata sovelma uudestaan palvelimelta asiakkaalle vähentävät Java-sovelman houkuttelevuutta muihin ratkaisuvaihtoehtoihin nähden.

Palvelun käyttäjän näkökulmasta parhaimmiksi vaihtoehtoiksi erottuvat HTML5 ja WebRTC ja BitTorrent ratkaisut. Molemmissa ratkaisuissa erityisen hyvää on se, että palvelu toimii sellaisenaan ilman interaktiota käyttäjältä. Käyttäjän näkökulmasta molemmissa tapauksissa palvelu tarjoaa täsmälleen saman toiminnallisuuden kuin mitä voisi olettaa puhtaassa asiakas-palvelin –mallissa. Huonoiten vertailussa puolestaan pärjäävät vaihtoehdot, jotka vaativat käyttäjältä asennuksia ja/tai jatkuvia ilmoitusten hyväksymisiä. Kaikkein huonoimmaksi vaihtoehdoksi valikoitui myös käyttäjänäkö-kulmassa Java-sovelma.

Kun palveluntarjoajan ja palvelun käyttäjän näkökulmat yhdistää, huonoimman tuloksen saa molempien vertailujen huonoin vaihtoehto eli Java-sovelma. Sen voinee siis suosiolla unohtaa saman tien. Parhaiten puolestaan pärjäsivät HTML5 ja WebRTC ja BitTorrent. Molemmat pärjäsivät itse asiassa parhaiten molempien näkökulmien suhteen, joten nämä vaihtoehdot ovat molemmat tarkemman harkinnan arvoisia. Koska HTML5 toteutus ei ollut mahdollinen valituksi tulee WebRTC ja BitTorrent yhdistelmä. Seuraavassa luvussa käydään läpi valitun ratkaisun toteutusta.

## 4 Koodaus

Koodi pohjautuu täysin Feross Aboukhadijehin aloittamaan avoimen lähdekoodin projektiin nimeltä Webtorrent [Web16] [Abo16]. Gradun kirjoittamisen alussa Webtorrent projektin oli vielä täysin suunnitteluasteella, mutta se kehittyi toimivaksi konseptiksi. Gradun koodausvaiheen pääasiallisiksi vaiheiksi jäi lopulta vain pienten muutosten teko valmiiseen koodiin.

### 4.1 Koodaus

Webtorrent projektin kunnianhimoisena tavoitteena on olla uusi laajennos BitTorrent-protokollaan siten, että BitTorrent toimisi täysin myös Internet-selaimissa. Sivutuotteena projektissa on luotu demo videovuon toistamisesta verkon ylitse. Tuo videovuo muodostetaan BitTorrent-protokollalla muilta asiakkailta haetuista palasista. Eli kyseisessä demossa saavutetaan juuri se tavoite, jota tässä gradussa tavoiteltiin.

Koodauksen vaiheet olivat

#### Valitun toteutuskielen oppiminen

Webtorrent on koodattu Node.js kielellä (<https://nodejs.org/>) eli Javascriptin yhdellä

muodolla. Vaikka Javascript oli entuudestaan tuttua, niin node.js toi mukanaan omia erityispiirteitään, jotka oli pakko opetella.

### **Koodin sisällön ymmärtäminen**

Samoin kuin kielen oppimisessa oli myös olemassaolevan projektin pariin siirtymisessä omat haasteensa. Projektin kehittäjät noudattavat onneksi tiukasti standardeja tapoja tehdä asiat. Node.js koodaustyyliin kuuluvat useat pienet osakokonaisuudet, moduulit, jotka tekevät jonkin asian hyvin ja eivät mitään ylimääräistä. Täten projektin ymmärtäminen vaatiikin ensisijaisesti sen ymmärtämistä, miten eri moduulit toimivat keskenään.

### **Koodausympäristön ja -työkalujen asennus**

Koodaus tapahtui Ubuntu Linux ympäristössä. Node.js koodaus vaatii omien komentoriviympäristön välineiden asennuksensa. Itse koodausta voi tehdä vaikka tekstieditorilla mutta käytön helppouden takia ohjelmointiympäristöksi valittiin yksi Eclipse ohjelmointiympäristön liitännäisistä Nodeclipse.

### **Koodimuutosten teko**

Koodimuutokset käytännössä valmiiseen projektiin olivat hyvin pieniä. Käytännössä kaikki muutokset liittyivät siihen, että testaus saataisiin suoritettua. Oli mm. tärkeää saada kaikki viitteet Internet-palvelimiin poistettua, jotta testauksen voi suorittaa suljetussa sisäverkossa.

### **Testausympäristön pystytys**

Testiympäristöä varten piti haalia kasaan useita eri tietokoneita ja miettiä miten ne liitetään toisiinsa verkoksi. Ohjelmallisella puolella käytettiin hyväksi koodausympäristön pystytysvaiheen asennuksia. Node.js koodia pystyi suorittamaan suoraan komentoriviltä. Testauksesta on oma erillinen osionsa tässä gradussa.

## **4.2 Koodin kuvaus**

Koodi pohjautuu avoimen lähdekoodin projektiin nimeltä Webtorrent. Webtorrent pohjautuu BitTorrenttiin ja WebRTC:hen ja pyrkii tuomaan BitTorrentin toiminnallisuuden suoraan selaimessa toimivaksi. Se on koodattu Javascriptillä ja toimii siksi periaatteessa missä tahansa Internet-selaimessa. Rajoituksia kuitenkin asettaa se, että sen tarvitsemat WebRTC-standardit ei ole vielä kovin hyvin tuettuja kaikissa selaimissa ja toisaalta myös se, että eri selainten kehittäjät tulkitsevat standardia eri tavoin. Webtorrentin toteu-

tus sisältää sekä seurantal palvelimen että asiakaspään toteutuksen BitTorrent-protokollasta websocket-yhteyksiä käyttäen. Webtorrent tukee sekä normaaleja BitTorrent yhteyksiä että omia WebRTC-yhteyksiään. Jos Webtorrenttia kuitenkin käyttää netiselaimessa, niin asiakas on rajoittunut yhteyksissään puhtaasti WebRTC-yhteyksiin, koska selainten tietoturvarajoitteista johtuen vain WebRTCn kautta sallitaan ns. cross-domain yhteydet. Saman alkuperän käytäntö sanelee sen, että sisältöä ei voida ohjelmallisesti hakea muista lähteistä, kuin sieltä, mistä Internet-sivusto on selaimeen ladattu. WebRTC kuitenkin sallii yhteydet muihin asiakaskoneisiin jo ihan perusominaisuutenaan, joten sen kautta tietoa voidaan hakea myös muualta. Webtorrent tukee (sekä seurantal palvelin että asiakaspään toteutuksessaan) yhteyksiä myös tavallisiin BitTorrent-asiakasohjelmiin. Se toimii siis ns. hybridinä näiden kahden eri kommunikointitavan välillä. Jotta Webtorrent asiakasta voisi käyttää hybriditilassa, sitä täytyy ajaa työpöytäohjelmana Internet-selaimen ulkopuolella. Webtorrent asiakasohjelma ei voi myöskään ottaa yhteyttä tavallisiin BitTorrent-asiakasohjelmiin käyttäen WebRTC-protokollaa. Tämä johtuu siitä että tavalliset BitTorrent-asiakasohjelmat eivät tue websocket-protokollaa vaan ovat rajoittuneita datan siirtoon TCP- ja UDP-protokollilla.

#### 4.2.1 WebRTC ja BitTorrent

WebRTCn ja BitTorrentin yhdistelmää on kuvattu jo tarkemmin luvussa 3.7. Webtorrent projekti sisältää yhden toteutuksen tästä. Kun ensimmäinen asiakas saapuu videon sisältävälle sivustolle alkaa tämä lataamaan videotiedostoa palvelimelta. Kun tämä bootstrap vaihe on suoritettu, niin seuraava sivustolle saapuva asiakas käyttääkin sitten jo BitTorrenttia ladatakseen tiedoston muilta käyttäjiltä. Tässä noudatetaan siis selkeästi mukautuvan sisällönjakelun ja vertaisavusteisen sisällönjakelun periaatteita.

WebRTC-protokolla tarjoaa mahdollisuuden suoratoistaa videodataa käyttäjältä toiselle (MediaStream Processing API), mutta Webtorrent ei hyödynnä tätä. Tälle on nähdäkseni kaksi pääsyytä. Ensinnäkin Webtorrent pyrkii tarjoamaan mahdollisuuden kaiken datan siirtoon selaimesta toiseen, ei vain videodatan (joka on lisäksi rajattu pieneen määrään mahdollisia videoenkoodaustyyppejä). Toiseksi MediaStream Processing API on tarkoitettu korkeamman tason video- ja audiovirroille kun taas BitTorrent-protokollalla siirretään matalammalla tasolla dataa pienempinä palasina. Edellä mainituista syistä johtuen Webtorrent hyödyntääkin kaikkein perustavimman laatuista WebRTC mekanismia eli tietokanavia (data channel) [Ris14].



WebRTC tarjoaa matalan tason tiedonsiirtokanavan, jota BitTorrent sen päällä sitten hyödyntää. Asiakaskoneet ottavat yhteyden seurantal palvelimeen websocket-protokollalla käyttäen BitTorrent-protokollan normaaleja kutsuja. Saatuaan tiedot seurantal palvelimelta BitTorrent-asiakas päättää keihin muihin asiakaskoneisiin se ottaa yhteyden. Yhteys muodostetaan WebRTC:tä käyttäen ja datavirta liikkuu BitTorrent-protokollan sanelemin säännöin asiakaskoneiden välillä. Koska tarkoituksena on tuottaa asiakkaalle katsottavaa videota, käytetään lisäksi BitTorrent-protokollan laajennosta, joka tuottaa mahdollisimman järjestyksessä olevaa datavirtaa.

## 4.2.2 Videon toisto

Videodatavirta saapuu asiakaskoneelle WebRTC ja BitTorrent –yhdistelmällä ja se putkitetaan eteenpäin HTML5 videoistimeen. Samoin mahdollistuvat aikasiirtymät tiedoston sisällä suoraan videoistimesta käsin. Jos käyttäjä siirtyy videossa eri kohtaan, kertoo videoistin eteenpäin tarpeestaan saada (kenties vielä lataamatonta) dataa eri kohdasta videovuota. Jos käytetty internetselain ei tue MSE:tä, ladataan tiedosto ensin kokonaan valmiiksi taustalla, ennen kuin se voidaan toistaa paikallisena tiedostona.

## 4.2.3 Esimerkit

Liitteessä 1 on esimerkki käytetystä node.js koodista. Koodista näkyy hyvin se, että toteutus on jaoteltu pieniin moduulikokonaisuuksiin ja näinollen koodin suoritus redusoituu käytettyjen moduulien määrittelyyn ja niiden käyttöönottoon.

## 4.2.4 Ongelmat

Koodi luottaa vahvasti siihen, että WebRTC ja MSE ovat tuettuja ominaisuuksia käytetyssä Internet-selaimessa. Jos jompaakumpaa ei tueta, ei videota pystytä hakemaan ja näyttämään halutulla tavalla. Esimerkiksi Internet Explorer ei tue kunnolla kumpaakaan ja näin ollen videota ei pystytä hakemaan eikä näyttämään. Sellaisia selaimia varten, jotka pystyvät näyttämään videon, mutta eivät hakemaan sitä WebRTC:tä käyttäen on olemassa fallback mahdollisuus, jossa videota haetaan suoraan palvelimelta. Tämä sama ominaisuus toimii myös ns. bootstrappina eli videon pitää olla olemassa vähintään yhdellä asiakkaalla, jotta sitä voisi alkaa jakamaan asiakkaalta toiselle. Jos taas selain tukee WebRTC:tä, niin video pystytään kyllä hakemaan ja sitä voidaan jakaa eteenpäin muille käyttäjille, mutta sitä ei pystytä välttämättä näyttämään selaimessa itsessään käyttäjälle.

## 5 Testaus

Testausosiossa käydään läpi valitun lähestymistavan yksi toteutus yksinkertaistetussa ympäristössä lähiverkossa. Suurin ero todelliseen reaali maailman ympäristöön on välilaitteiden puuttuminen. Tällaisia välilaitteita voisivat olla esimerkiksi palomuurit, NAT-laitteet ja välityspalvelimet. Testauksen pääasiallinen tutkimuskohde on selvittää toimii-ko tutkittava järjestelmä teorian mukaisesti.

### 5.1 Testauksen toteutus

Testauksessa oli käytössä viisi tietokonetta lähiverkossa. Näistä koneista neljä oli testeissä asiakaskoneina ja yksi palvelinkoneena. Koetilanteen tasalaatuisuuden vuoksi jokainen asiakaskone käyttää samaa Internet-selainta ja sen uusinta versiota. Pääasiallisena muutoksena eri testiskenaarioiden välillä on se, kuinka moni asiakaskone on jakamassa videotiedostoa ja kuinka moni on lataamassa sitä. Samat testiskenaariot toistetaan siten että solmujen nettiyhteysnopeudet on rajoitettu eri nopeuksiksi. Nettiyhteyttä rajoitettiin aina vain niissä solmuissa, joista data oli kulkemassa eteenpäin eli ns. vastaanottajapään kaistankäyttöä ei rajoitettu. Yhteysnopeuksien rajoitukseen on päädytty kahdesta syystä. On hyvä ymmärtää, että tosielämässä asiakaskoneiden Internet-yhteydet ovat paljon rajoitetumpia kuin palvelinkoneiden yhteydet. Ja toisaalta ilman rajoituksia testattavat nopeudet nousevat niin suuriksi, että eri skenaarioiden keskinäinen vertailu hankaloituu tai ei ole mielekästä.

#### 5.1.1 Mitä testataan

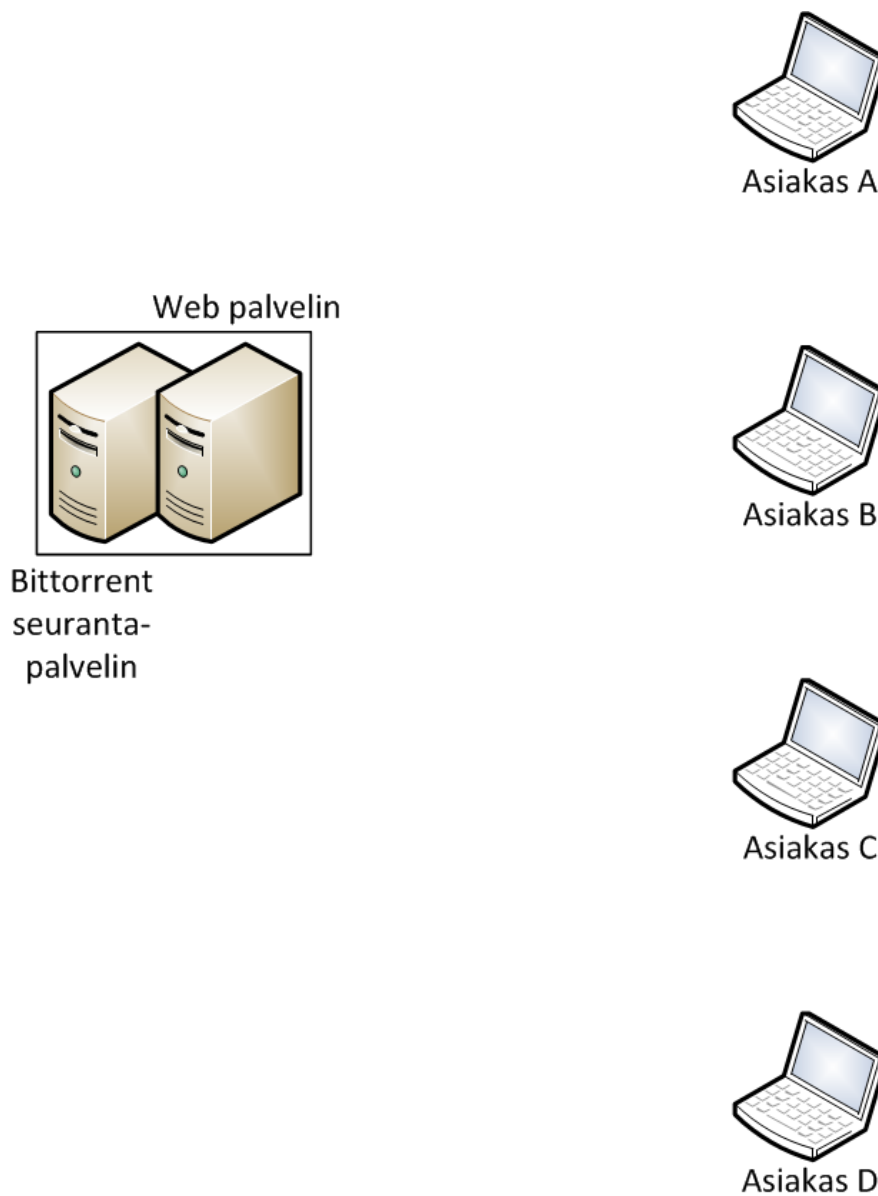
Tarkoituksena on käytännössä selvittää toimiiko järjestelmä halutusti eli pystyykö videotiedosto siirtymään asiakkaiden välillä ilman palvelinta. Lisäksi tutkitaan toimiiko järjestelmä oletusten mukaisesti, kun jakavien asiakkaiden määrää muutellaan tai kun lataavien asiakkaiden määrää muutellaan.

#### 5.1.2 Testattava järjestelmä

Asiakaskoneilla oli asennettuna Microsoft Windows ja palvelinkoneella Ubuntu Linux. Palvelimella ei simuloinnissa haluttu olevan muuta roolia, jotta nettiliikenne oikeasti kulkisi verkossa koneelta toiselle ja sitä olisi selkeämpi seurata. Asiakaspään koneissa asiakasohjelmana käytettiin Google Chromea. Chromen WebRTC tuki on tällä hetkellä yksi parhaimmista ja se sopii muutenkin hyvin testaukseen, koska se pystyy suoratois-

tamaan ladattua videota selaimessa suoraan.

Testauksessa kaikki verkon solmut ovat yhden reitittimen kautta luodussa lähiverkossa. Palvelinkone on kiinni reitittimessä yhden gigan ethernet-liitännällä ja muut solmut ovat WLAN-yhteydessä käyttäen 802.11n standardin mukaista nopeutta. Palvelinkone toimii videonjakopalvelimena, joka tarjoaa ulospäin html-muotoista nettisivustoa. Sivusto tarjoaa html-sisällön lisäksi ulospäin yhtä videotiedostoa sekä javascript tiedostoa, joka asiakaspäähän ladattuna toimii asiakaspään ohjelmistona. Tämä samainen kone toimii myös BitTorrent seurantapalvelimena. Jaettavan videotiedoston koko on 97,8 megatavua (tarkemmin 102 638 434 tavua), jonka voidaan todeta olevan riittävän iso simuloidakseen reaali maailman tilannetta lyhyestä videotiedostosta. Muut koneet toimivat samassa lähiverkossa asiakaskoneina. Kuvassa 4 on kuvattuna perus testiasetus. Kuvasta on jätetty yksinkertaisuuden vuoksi reititin pois, mutta kaikki liikenne kulkee luonnollisesti sen kautta.



Kuva 4. Testattavan järjestelmän osakomponentit.

### 5.1.3 Skenaariot

Testauksessa käytetään viittä eri skenaariota. Eri skenaariot toistetaan neljä kertaa siten, että jokaisella kerralla koneiden nettiyhteyksnopeutta on rajoitettu eri tavoin. Yhteensä eri tapauksista taulukoidaan siis  $5 * 4 = 20$  lukuarvoa. Vertailussa mitataan myös mitä nopeutta tiedosto siirtyy palvelimelta asiakaskoneelle tavallisella TCP-yhteydellä ilman rajoituksia.

#### Skenaario 1

Lähteiden määrä: 1

Lataajien määrä: 3

### **Skenaario 2**

Lähteiden määrä: 1

Lataajien määrä: 2

### **Skenaario 3**

Lähteiden määrä: 1

Lataajien määrä: 1

### **Skenaario 4**

Lähteiden määrä: 2

Lataajien määrä: 1

### **Skenaario 5**

Lähteiden määrä: 3

Lataajien määrä: 1

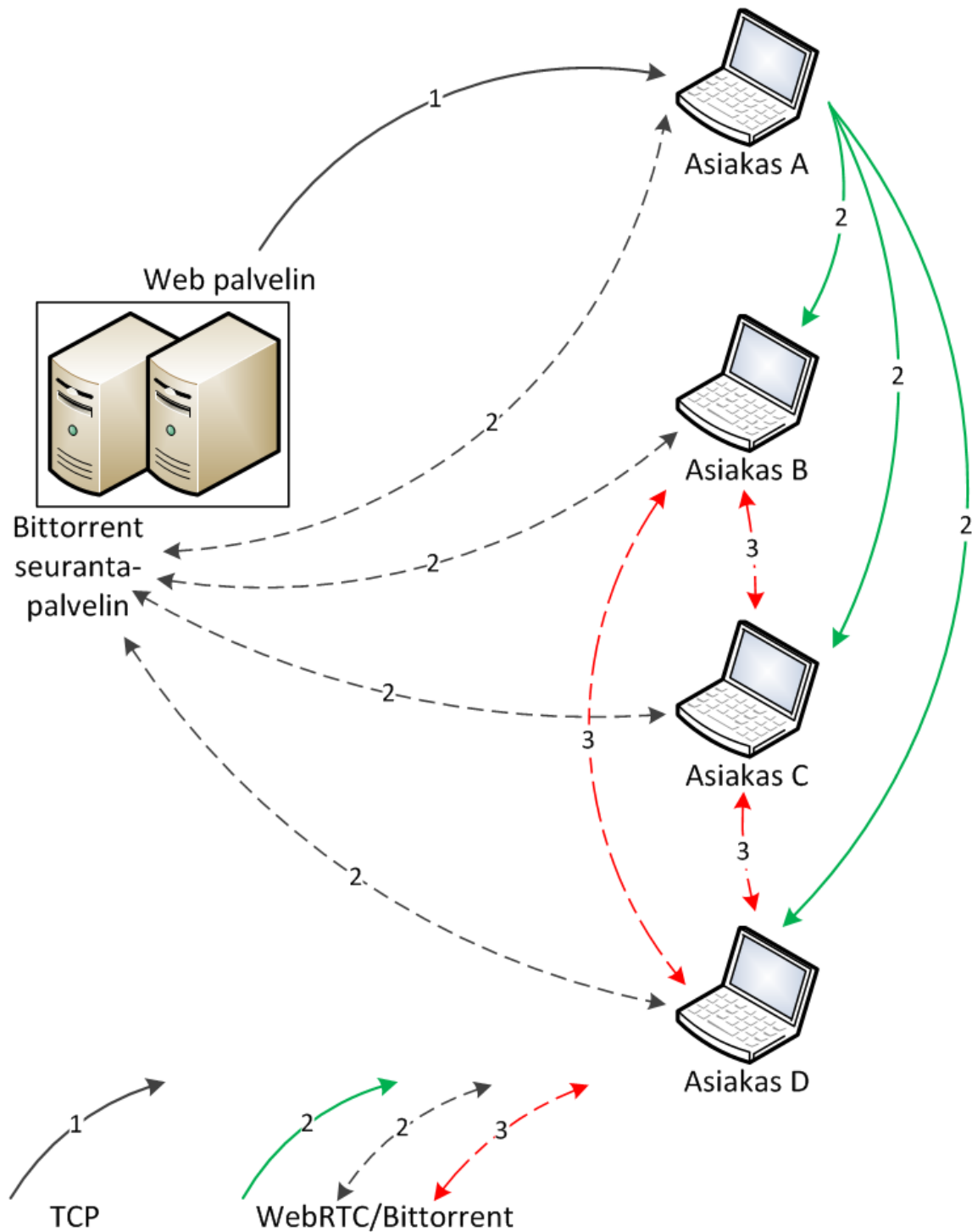
Seuraavassa on kuvattu yksi testauskierros (Skenaario 1, nopeus rajoitettu 50/50) yksityiskohtaisesti. Kierroksen eri vaiheet on kuvattu Kuvassa 5:

1. Videonjakopalvelu käynnistetään palvelimella. Samassa IP-osoitteessa, mutta eri porteissa, vastaa tämän jälkeen sekä web-palvelin että WebRTC/BitTorrent seurantapalvelin.
2. Yhden asiakaskoneen (Asiakas A) Internet-selaimesta (Google Chrome) avataan yksi välilehti yksityisyystilassa
3. Asiakas A:n selaimen välilehteen avataan videonjakopalvelun sivusto ja selain lataa html-sivun lisäksi automaattisesti videon kokonaisuudessaan käyttäen normaalia TCP-yhteyttä (Kuvan 5 vaihe 1). Tässä on kyse ns. bootstrapista eli videon pitää olla olemassa vähintään yhdellä asiakkaalla, jotta sitä voisi alkaa jakamaan asiakkaalta toiselle.
4. Kun videotiedosto on kokonaan ladattu, poistetaan se videonjakopalvelun jakokansioista kokonaan pois. Tämä tehdään siksi, että voidaan olla varmoja, että

tästädes ainut kopio videotiedostosta on olemassa vain asiakaskoneilla. Normaalissa videonjakopalvelussa mahdollisuus uuteen bootstrappiin säilytettäisiin toki aina.

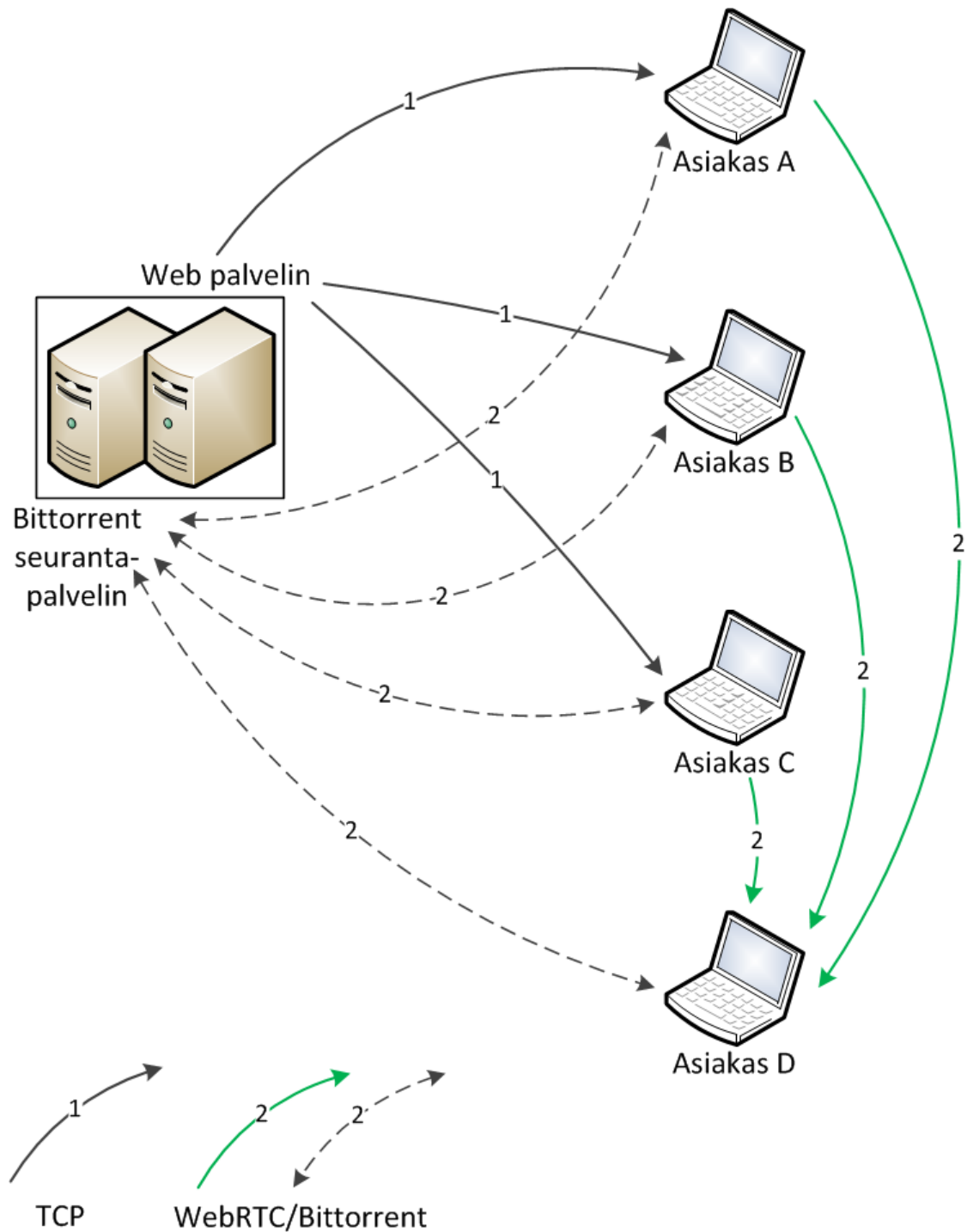
5. Asiakas A:n Chrome-selaimen internetnopeutta rajoitetaan siten että maksimi lähetysnopeus on 50 kilotavua sekunnissa ja maksimi latausnopeus on 50 kilotavua sekunnissa
6. Kolmen muun asiakaskoneen (Asiakas B, Asiakas C ja Asiakas D) Internet-selaimesta (Google Chrome) avataan yksi välilehti yksityisyystilassa
7. Asiakkaiden A, B ja C selaimen välilehteen avataan videonjakopalvelun sivusto ja selain alkaa lataamaan automaattisesti videotiedostoa asiakkaalta A käyttäen WebRTC/BitTorrent-protokollaa (Kuvan 5 vaihe 2).
8. Kun asiakkaat B, C ja D saavat osia jaettavasta tiedostosta alkavat ne samalla jakamaan niitä eteenpäin muille lataajille (Kuvan 5 vaihe 3).
9. Videotiedoston kokonaissiirtymisaika asiakkaalta A asiakkaille B, C ja D mitataan ja samalla tarkkaillaan nettiliikennettä asiakaskoneilla ja videonjakopalvelun päässä
10. Koko testiasetelma puretaan. Palvelinohjelmisto ajetaan alas ja Chrome-selaimet suljetaan. Koska kutakin selainta on käytetty yksityisyystilassa, tämä varmistaa että kaikki testauskierrokseen liittyvä data poistetaan.

Skenaarion yksi pitäisi teoriassa tuottaa kaikkein hitaimmat kokonaissiirtoajat. Tämä johtuu siitä, että jakajien määrä on pienin mahdollinen ja lataajien määrä on suurin mahdollinen (testiasetelman puitteissa).



Kuva 5. Testiskenaario, jossa lähteitä on yksi kappale ja lataajia kolme.

Kuvassa 6 nähdään kuvattuna Skenaario 5. Tässä testiskenaariossa jakajia on (testiasetelman puitteissa) suurin mahdollinen määrä ja lataajia pienin mahdollinen määrä. Tämän skenaarion pitäisi näin ollen tuottaa teoriassa kaikkein nopeimmat siirtoajat.



Kuva 6. Testiskenaario, jossa lähteitä on kolme kappaletta ja lataajia yksi

Teoriassa, jos kaikki häiriötekijät jätettäisiin huomiotta, pitäisi kussakin skenaariossa siirtonopeuksien pienentyä isompaan skenaarionumeroon siirryttäessä. Tämä sama no-  
peutuminen pitäisi näkyä kaikissa eri nopeusluokissa. Skenaariossa 3 on lähteiden ja  
lataajien lukumäärä 1, joka vastaa eniten normaalia asiakas-palvelin –mallin tilannetta.



## 5.2 Tulokset

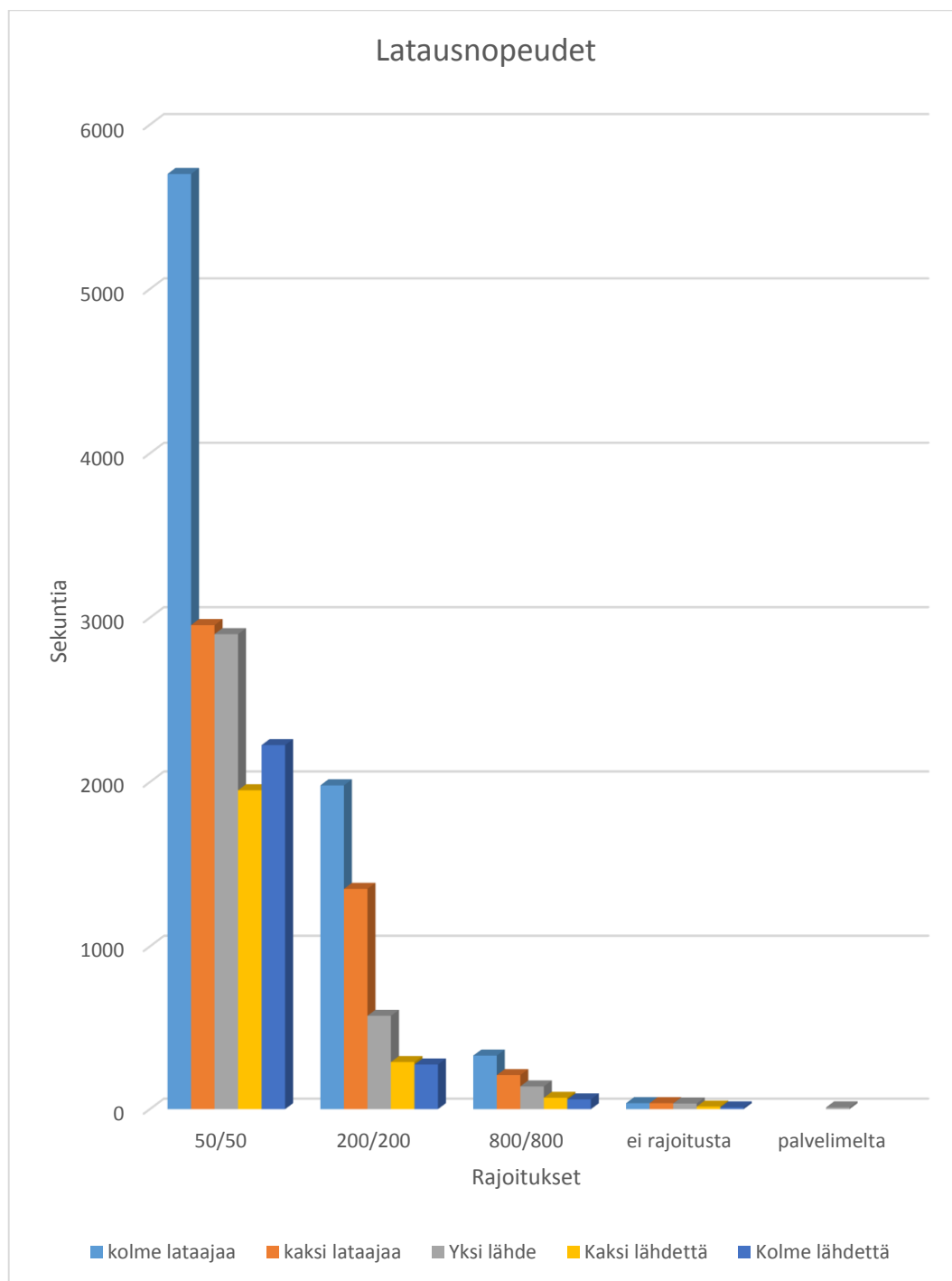
Testaus onnistui hyvin ja kaikista skenaarioista saatiin lukuarvo ulos. Kaikissa tapauksissa testiä toistettiin vähintään kahdesti sen varmistamiseksi, että saatu tulos ei ollut sattumaa. Epäselvissä tapauksissa testejä toistettiin vielä tätäkin useampia kertoja. Tuloksien listauksissa on otettu vertailukohdaksi myös teoreettisia arvoja siirroille, jotta saadaan vertailukohtaa teorian ja testauksen tuloksen välille.

Taulukossa 2 on listattuina kaikki arvot, jotka testauksista saatiin. Kussakin taulukon solussa on se aika sekunteina, joka tiedoston siirrosta kului kunkin skenaarion kaikille lataajille. Jos lataajia oli useampia, aloitettiin lataus kaikilla lataajilla samanaikaisesti. Taulukon solun lukuarvo kertoo tällöin kauanko koko siirto kesti.

	<b>50/50</b>	<b>200/200</b>	<b>800/800</b>	<b>ei rajoitusta</b>	<b>palvelimelta rajoituksetta</b>
<b>Kolme lataajaa</b>	5700	1980	330	37	
<b>Kaksi lataajaa</b>	2955	1350	210	37	
<b>Yksi lähde</b>	2900	575	140	35	11
<b>Kaksi lähdettä</b>	1950	290	70	16	
<b>Kolme lähdettä</b>	2225	275	60	11	

*Taulukko 2. Testauksen tulokset taulukkona.*

Kaaviossa 1 ovat Taulukon 2 tulokset graafisen kuvaajan muodossa. Kuvaajan avulla nähdään ehkä paremmin miten tulokset keskenään sijoittuivat sekä saman skenaarion sisällä että skenaarioiden välillä. Selkeä trendi on nähtävissä molemmissa katsantokannoissa. Mitä useampia jakajia videolla on, sitä nopeammin se siirtyy lataajille. Samoin se on selkeää, että mitä suuremmiksi lataus- ja jakonopeuksien annetaan nousta, sitä nopeammin video siirtyy lataajille. Vaikka trendi onkin selkeä, niin yksittäiset tulokset eivät noudata oletettua kaavaa orjallisesti. Tämän syitä pohditaan enemmän luvuissa 5.2.1 ja 5.2.2.



*Kaavio 1. Testauksen tulokset kaaviona.*

Data siirtyy 1367 tavun kokoisina UDP-sanomina, josta todellista sovellusdataa on 1312 tavua. Paluuviestien koko on 135 tavua ja paluuviestejä on yhtä monta kuin datasanomia. Lisäksi solmujen välillä liikkuu hyvin paljon muita UDP-sanomia, jotka liittyvät STUN-viestinvälitykseen.

### 5.2.1 Tarkempaa tarkastelua (Skenaario 1)

Hitaimmalla nopeudella (50kt/50kt) todella suuri osa liikenteestä tuntui olevan ylimääräistä yleiskustannusta. Paluuliikenteen kaista oli suuressa käytössä koko siirron ajan ja vaikutti siltä, että se osaltaan myös hidasti selkeästi siirtoa. Jos paluuliikenteen kaistaa ei olisi rajattu ollenkaan, on mahdollista, että itse datan siirtonopeus olisi ollut parempi.

Nopeudella 200kt/200kt todellinen lähetysnopeus lähettäjäkoneesta käsin vaikutti olevan nyt noin 200kt/s. Latausnopeus puolestaan oli noin 70kt/s. Sen jälkeen kun siirto oli tehty, niin edelleen viestejä liikkui jatkuvasti solmujen välillä nopeuksilla 60kt/s sisään ja 50kt/s ulos (lähettäjäkoneelta). Tämä liikenne tuntui olevan puhtaasti reititysviestejä. Verkko analysaattorista (Wireshark) näkyy (Kuva 7), että tutkittu solmu (IP: 192.168.1.236) on jatkuvassa yhteydessä sisäverkon muihin koneisiin.

yllapito\_cap\_example.pcapng [Wireshark 1.12.1 (v1.12.1-0-g01b65bf from master-1.12)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply Save

No.	Source	Destination	Protocol	Length	Info
109	192.168.1.190	192.168.1.236	STUN	154	Binding Request user: 6Eu1vccCH6NgLHso:fBaev
110	192.168.1.190	192.168.1.236	STUN	154	Binding Request user: 0r50j6zuUwVa3jerb:kig7N
111	192.168.1.190	192.168.1.236	STUN	158	Binding Request user: 16rwxJ84gjrQjw/w:2AGPt
112	192.168.1.236	192.168.1.29	STUN	154	Binding Request user: wgQLZLkKZmTd989v:wm3HY
113	192.168.1.29	192.168.1.236	STUN	154	Binding Request user: 3zJFaTu0leik1Pco:jME4t
114	192.168.1.190	192.168.1.236	STUN	158	Binding Request user: jOFVny6wkSr0jNOY:HkAEEm
115	192.168.1.236	192.168.1.190	STUN	154	Binding Request user: vF2L0d0Vb9x1r1lv:+PdTx
116	192.168.1.236	192.168.1.29	STUN	154	Binding Request user: 75tLU4xr04ffut9l:1mc96
117	192.168.1.236	192.168.1.190	STUN	154	Binding Request user: aMnz8iO+oks/w05w:9Yxpx
118	192.168.1.236	192.168.1.29	STUN	158	Binding Request user: 8tP6xTdNEexs6dUN:s1ptc
119	192.168.1.236	192.168.1.190	STUN	158	Binding Request user: kig7NSWpcPASN8rv:Or50j
120	192.168.1.236	192.168.1.29	STUN	154	Binding Request user: 9lKNeKKT5mAHvny:01jFi
121	192.168.1.236	192.168.1.29	STUN	154	Binding Request user: G4+EQZ1eWRNdQA5s:0oGBE
122	192.168.1.236	192.168.1.190	STUN	154	Binding Request user: KvBBavxndsCl9DXC:bGTVf
123	192.168.1.236	192.168.1.29	STUN	154	Binding Request user: eAnzfPpSE4rvv+23:FZMz+
124	192.168.1.236	192.168.1.29	STUN	158	Binding Request user: 6mp6G8oyKk1kuxst:0H4Gy
125	192.168.1.236	192.168.1.190	STUN	158	Binding Request user: S7AFn8r/zD0s1f1c:1Dm9q
126	192.168.1.236	192.168.1.190	STUN	154	Binding Request user: dP9wxGFNikmKA/tg:QOACv
127	192.168.1.29	192.168.1.236	STUN	154	Binding Request user: mFFEeAF5xkp3Boat:Sero0
128	192.168.1.29	192.168.1.236	STUN	158	Binding Request user: eE5MQASyUiy+Dn/A:TGLEQ
129	192.168.1.236	192.168.1.29	STUN	154	Binding Request user: rZeRnboOLD96tY6S:zoIKO
130	192.168.1.236	192.168.1.190	STUN	154	Binding Request user: 1ySqaE04KJ+GYi5k:px3aQ
131	192.168.1.236	192.168.1.190	STUN	158	Binding Request user: Ybn/GHIjw0H/bPuy:B3UGT
132	192.168.1.190	192.168.1.236	STUN	154	Binding Request user: g5h3Fe+xbDngFbea:A57Qb
133	192.168.1.190	192.168.1.236	STUN	154	Binding Request user: o3qTEZKg6pwZjVqG:bsPV/
134	192.168.1.236	192.168.1.29	STUN	158	Binding Request user: YRM1YghjgvKB2er1:LrfeL
135	192.168.1.236	192.168.1.190	STUN	154	Binding Request user: yAEZHAH12/3Flhyp:vEsLZ
136	192.168.1.236	192.168.1.29	STUN	158	Binding Request user: UJdMniIU8E8bqbqv:IJCcj
137	192.168.1.236	192.168.1.190	STUN	158	Binding Request user: zJ4fegZg7rOI358B:8g5C3
138	192.168.1.236	192.168.1.29	STUN	154	Binding Request user: mIih7M4Q+8Dy4heF:Id0ob
139	192.168.1.236	192.168.1.29	STUN	154	Binding Request user: nQ/K9E3jfk2BwBsn:sgwr3
140	192.168.1.236	192.168.1.190	STUN	154	Binding Request user: mc9LLL2jwtkJ31EL:VxVL6
141	192.168.1.236	192.168.1.190	STUN	158	Binding Request user: 8Y4sep0E19z1untz:bv/lu
142	192.168.1.29	192.168.1.236	STUN	154	Binding Request user: wxQZwtGrnwPVudE3:Lw5Kg
143	192.168.1.29	192.168.1.236	STUN	158	Binding Request user: ENpsPtmi206JDX9C:M4GdF
144	192.168.1.236	192.168.1.29	STUN	158	Binding Request user: oCBXS9DK4fxd/uiH:E1oAB
145	192.168.1.236	192.168.1.190	STUN	154	Binding Request user: UDDwqnvKV+IE+6Lo:EEdzC
146	192.168.1.236	192.168.1.29	STUN	154	Binding Request user: 2ra41f+qWFltBzpv:tnmdm
147	192.168.1.236	192.168.1.29	STUN	158	Binding Request user: PbsF0zuyk9DRSulr:Etxyf
148	192.168.1.236	192.168.1.190	STUN	158	Binding Request user: bsPV/i2ieAlD/bta:o3qTE
149	192.168.1.236	192.168.1.190	STUN	154	Binding Request user: nQanWUarac0i9tk:ntthc

0000 c8 60 00 94 2e a5 5c 51 4f d6 04 66 08 00 45 00 .\....\Q 0..f..E.  
0010 00 30 6d b0 00 00 80 11 5c ea c0 a8 01 ec 17 15 .0m.....\.....

File: "E:\gradu\testit2\lappari\yllapito\_cap\_e... Packets: 193999 · Displayed: 193999 (... Profile: Default

Kuva 7. Datan siirron jälkeen sisäverkon solmu (IP: 192.168.1.236) on edelleen yhteydessä muihin koneisiin.

Nopeudella 800kt/800kt todellinen lähetysnopeus oli noin 700kt/s. Nopeus tuntui heittelevän todella paljon.

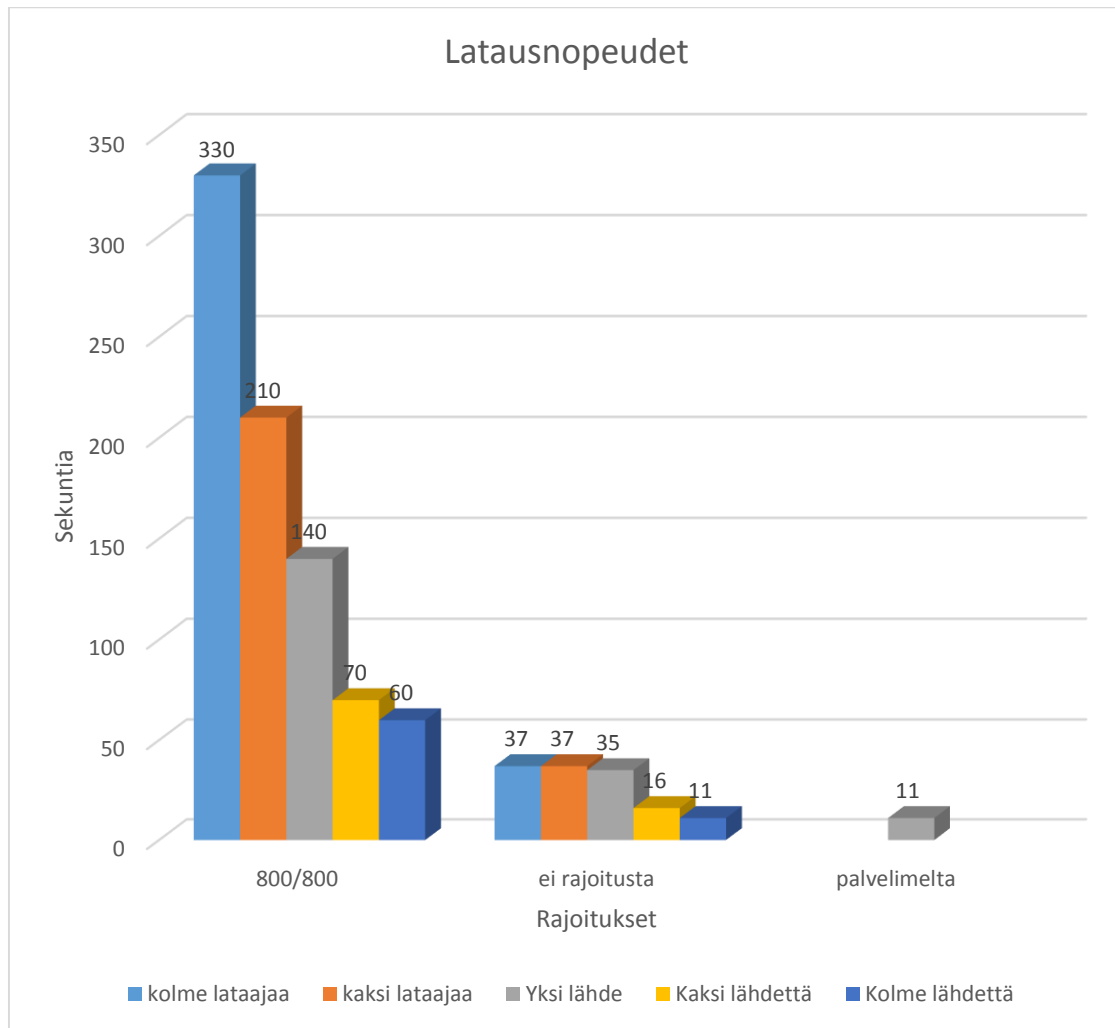
Tuloksissa kolmen lähteen skenaario ei noudattanut hitaissa nopeuksissa sitä kaavaa, jota sen olisi pitänyt noudattaa. Nopeammissa nopeuksissa, ja erityisesti täysin ilman rajoituksia tilanteessa, se kuitenkin taas oli täysin linjassa sen kanssa mitä sopi odottaa. Syitä tälle ei tutkittu tarkemmin, mutta joitakin valistuneita arvauksia voidaan tehdä. Hitaammilla nopeuksilla oli havaittavissa, että ylimääräiset yleiskustannukset veivätkäistää suhteessa enemmän kuin suuremmilla nopeuksilla. Kenties jopa niin, että hi-

taammilla nopeuksilla siirtonopeudet kärsivät oleellisesti siitä, että kaista oli jatkuvasti tukossa. Tästä samasta asiasta on viitteitä myös siinä, että hitaammilla nopeuksilla prosentuaalinen heitto teoreettisen tuloksen ja todellisen tuloksen välillä oli suurempi kuin suurilla nopeuksilla (Taulukko 4 ja Kaavio 4 luvun lopussa).

### **5.2.2 Tarkempaa tulosten tulkintaa**

Mittakaavan takia pienten lukujen keskinäinen järjestys ei näy Kaaviossa 1 riittävän tarkasti, joten Kaaviossa 2 esitellään kahden nopeimman nopeusluokan tuloksia paremmin. Kun rajoituksia on vähemmän, tai ne on poistettu kokonaan, eivät mahdolliset yleiskustannukset hidasta liikennettä enää yhtä paljoa. Kaista ei ole enää ”tukossa”. Lataajien määrän lisäyksen tuottama yleisliikenne ei rajoita enää hyötykuorman siirtonopeutta.

Kaaviosta näkyy myös se, että kun rajoitukset poistetaan, pystyy yksittäinen kone palvelemaan vaikeuksitta useampaa määrää muita koneita. Kun sekä lataajia että jakajia on yksi kappale, kestää videon siirto 35 sekuntia. Mutta vaikka lataajia lisättäisiin kahteen, kestää videon jako edelleen vain 37 sekuntia. Eikä kolmannen lataajan lisäys muuta tilannetta edelleenkaan mihinkään. Suoralla latauksella palvelimelta päästään kyllä nopeampaan siirtonopeuteen (11 sekuntia), mutta syyt tälle ovat jossain muualla.

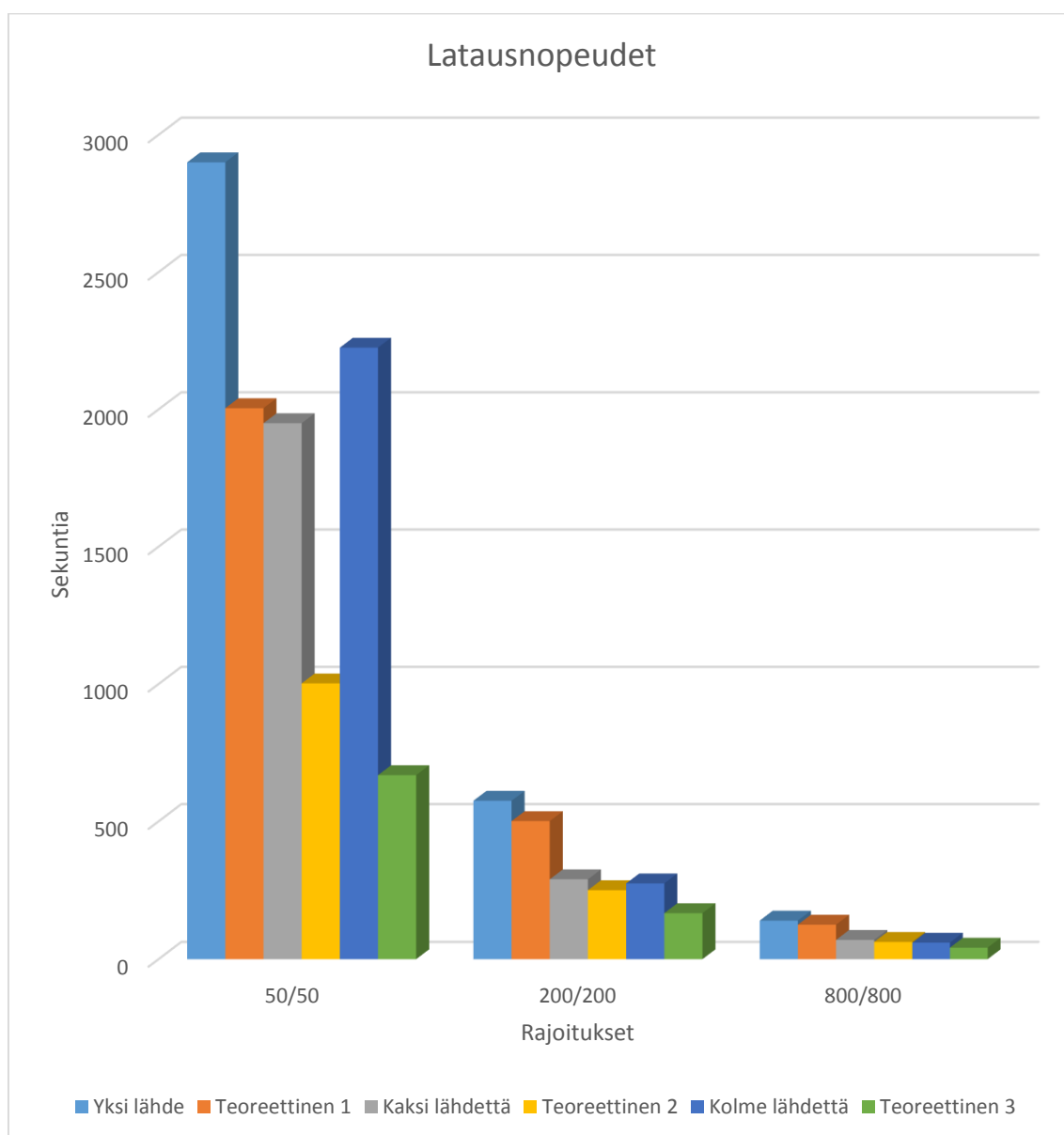


*Kaavio 2. Testauksen tulokset kaaviona. Tuloksista poistettu hitaimmat rajoitukset.*

Taulukossa 3 vertaillaan teoreettisten nopeuksien suhtautumista mitattuihin nopeuksiin. Näyttää olevan selvää, että mitatut nopeudet eivät yllä ihan samalle tasolle kuin teoreettiset, mutta selvää on myös se, että muutamaa hajatapausta lukuun ottamatta teoria ja tulokset kulkevat käsi kädessä. Kaaviossa 3 joka toinen palkki on mitattu arvo ja joka toinen on teoreettinen arvo. Rajoituksen ollessa 50/50 on nähtävillä selkeää epäjohtonukaisuutta mitatuissa tuloksissa, mutta suuremmissa nopeuksissa päästään jopa yllättävän lähellä teoreettisia arvoja.

	50/50	200/200	800/800	ei rajoitusta	palvelimelta
<b>Yksi lähde</b>	<b>2900</b>	<b>575</b>	<b>140</b>	<b>35</b>	<b>11</b>
<b>Teoreettinen 1</b>	<b>2005</b>	<b>501</b>	<b>125</b>		
<b>Kaksi lähdettä</b>	<b>1950</b>	<b>290</b>	<b>70</b>	<b>16</b>	
<b>Teoreettinen 2</b>	<b>1002</b>	<b>251</b>	<b>63</b>		
<b>Kolme lähdettä</b>	<b>2225</b>	<b>275</b>	<b>60</b>	<b>11</b>	
<b>Teoreettinen 3</b>	<b>668</b>	<b>167</b>	<b>42</b>		

*Taulukko 3. Testauksen tulokset taulukkona ja teoreettiset arvot niiltä osin kuin ne ovat helposti laskettavissa.*



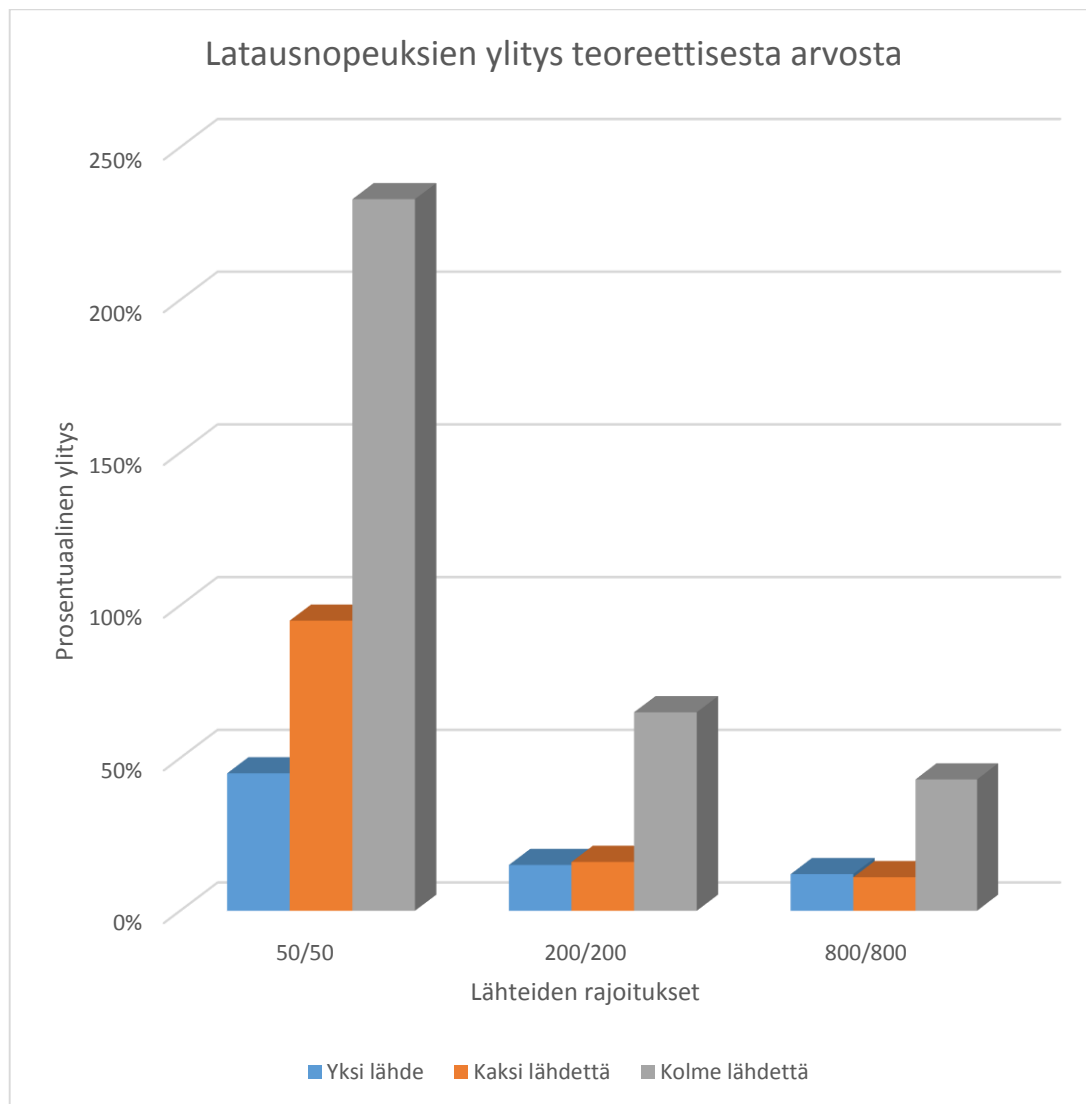
*Kaavio 3. Testauksen tulokset kaaviona teoreettisten arvojen kanssa. Teoreettiset arvot siltä osin kuin ne ovat helposti laskettavissa.*

Kuten jo aikaisemmin todettiin, mitä suurempi määrä solmuja testiskenaariossa oli, sitä suurempi määrä liikenteestä tuntui menevän yleiskustannuksiin. Tämä tulos näkyy myös, kun vertaillaan kuinka kaukana teoreettiset arvot ovat mitatuista arvoista (Taulukko 4). Tässä samoin tämän pystyy erityisesti toteamaan hitaammilla nopeuksilla. Skenaarioissa 1 ja 5 oli kaikkien eniten solmuja käytössä ja niissä molemmissa on havaittavissa suurta heittoa teoreettisen ja todellisen tuloksen välillä. Näistä kahdesta erityisesti skenaario, jossa jakajia oli eniten, tuntui aiheuttavan nopeuksien hidastumista (Kaavio 4).

	<b>50/50</b>	<b>200/200</b>	<b>800/800</b>
Yksi lähde	45 %	15 %	12 %
Kaksi lähdettä	95 %	16 %	11 %
Kolme lähdettä	233 %	65 %	43 %

*Taulukko 4. Todellisten latausnopeuksien ylitys teoreettisesta arvosta.*





Kaavio 4. Todellisten latausnopeuksien ylitys teoreettisesta arvosta.

Ainut selkeä poikkeava tulos datassa tuli hitaimmalla nopeudella Skenaariossa 5. Aiemmin mainittujen yleiskustannusten lisäksi tälle epäjohtonmukaiselle arvolle ei ole tiedossa syytä.

### 5.3 Yhteenveto

Yleisenä lopputuloksena testeistä voidaan todeta että teoria ja testaustulokset korreloivat ja järjestelmä toimii teorian mukaisesti. Muutamia poikkeavia havaintoja tehtiin ja niille löydettiin potentiaalisia syitä. Tarkempaa analyysia poikkeamista ei kuitenkaan tehty. Yleiskustannusten suuri määrä voi käytännössä selittyä kolmesta eri kustannuksista lisäävästä tekijästä tai näiden yhdistelmästä; WebRTCn signaointi, WebRTCn reititysviestit ja BitTorrent yleiskustannukset. Wiresharkin kautta tehty liikenteen

tarkkailu kertoi suuresta määrästä reititysviestejä, mutta on myös mahdollista että BitTorrentin käyttö itsessään [ShK06] on ongelman suurin syy.

## 6 Yhteenveto

Tässä tutkielmassa käytiin läpi hajautettua sisällönjakelua asiakaspään toteutuksena videonjakopalvelussa. Hajauttamalla videonjakelu käyttäjille vähennetään palvelimen kuormaa ja mahdolliset käyttäjäpiikit eivät estä palvelua toimimasta. Pienemmillä resursseilla varustettu taho pystyy toteuttamaan suurenkin palvelun.

Useaa eri mahdollista ratkaisua käytiin läpi ja vertailussa löydettiin kaksi tapaa, jotka erottuivat muista parempina; HTML5 sekä WebRTC ja BitTorrent -yhdistelmä. HTML5 ratkaisua ei ole mahdollista nykyisellään toteuttaa, mutta se olisi kenties ratkaisuna kaikkein paras. Tämä on toki vain teoriaa sillä moni asia jää arvailun varaan. WebRTC ja BitTorrent -yhdistelmä puolestaan on mahdollista toteuttaa ja pienessä mittakaavassa ratkaisu tehtiinkin ja sitä testattiin. Testien tuloksena voidaan todeta yksikantaan, että ratkaisun toimivuudesta on selkeää näyttöä.

Tutkielmassa käsitellyn WebRTC ja BitTorrent -yhdistelmän koodi on lähes täysin avoimen lähdekoodin projektin Webtorrentin tulosta. Webtorrent projektin kehitys on jatkunut koko tutkielman teon ajan ja se vaikuttaisi olevan hyvin lähellä pistettä, jossa se löisi itsensä läpi suuren yleisön käyttöön. Jos Webtorrentia haluaisi käyttää tämän tutkielman esittelemään tilanteeseen, pitäisi ratkaisussa olla kuitenkin vaihtoehtoinen tapa ladata video, jotta myös ei-tuetuilla selaimilla ja mobiililaitteilla pystyisi sivustoa käyttämään.

Puhdas HTML5 ei siis toimi sellaisenaan ja voi hyvin olla että ei koskaan tulekaan toimimaan. Saman alkuperän käytäntö on hyvin perustavaa laatua oleva rajoite ja sen poistaminen avaisi liikaa mahdollisuuksia haittaohjelmille. Toinen rajoite puolestaan voisi kin olla mahdollista poistaa. Asiakaspää voisi pystyä toimimaan palvelinpäänä, jos se vain mahdollistettaisiin. Toki tämänkin pitäisi tapahtua tietoturvarajoitusten puitteissa. Ei ole kuitenkaan välttämättä tarpeen tehdä kahtiajakoa HTML5:n ja WebRTC:n välillä. WebRTC toimii kuitenkin HTML5 ympäristössä ja ikään kuin vain lisää sen toiminnallisuutta. WebRTC on lisäksi W3C:n ja IETF:n alla kehitettävä standardi ja kenties tulevaisuudessa osa HTML5 kieltä.

## **6.1 Loppupäätelmät**

Tutkielman tulokset viittaavat siihen, että vain muutamaa palvelinta käyttäen pystytään rakentamaan hajautettu, skaalautuva videonjakopalvelu. Testit osoittivat, että itse idea toimii eli käyttäjien määrää kasvattamalla palvelun laatu ei heikkene vaan päinvastoin paranee. Testit koskivat toki vain pientä käyttäjämäärää, joten tarkemmasta skaalautuvuudesta suuremmille käyttäjämäärillä ei voida sanoa mitään tarkkaa.

Salamajoukon aiheuttama käyttäjäpiikki voi hetkellisesti lamauttaa palvelun, jos videotiedostosta ei ole riittävän montaa kopiota käyttäjillä. Tämä on seurausta ihan perus BitTorrent-protokollan käyttäytymisestä. Jos kuitenkin kopioita on riittävästi piikin iskiesä, voi käyttäjille hajautettu järjestelmä olla hyvinkin tehokas ratkaisu. Toisaalta siis suuri käyttäjämäärä voi tuottaa parempaa palvelua, mutta toisaalta, riippuen käyttäjäpiikin tarkasta muodostumistavasta, voi palvelun laatu olla heikkoakin. Palvelinkapasiteettia kasvattamatta saadaan kuitenkin siis aikaan järjestelmä jolla on ainakin teoreettiset mahdollisuudet toimia paremmin kuin perinteinen sisällönjakeluverkko.

## **6.2 Jatkotutkimuskohteita**

Jos tässä gradussa esiteltyä asiaa haluaisi tutkia vielä lisää, voisi yksi tutkimuskohde olla seurantapalvelimen tarpeen poisto ratkaisusta. BitTorrent puolelta tähän löytyy jo valmis ympäristö Hajautettu tiivistetaulu -tekniikan muodossa. Webtorrent ei pysty valmiita DHT-verkkoja hyödyntämään, koska selaimen päällä toimivat asiakkaat ovat sidottuja websocket-protokollaan. Mutta kenties tulevaisuudessa tulee websocket kykyisiä DHT-verkkoja.

Toinen mahdollinen tutkimuskohde voisi olla nettiliikenteen tarkempi analysointi. Kuinka paljon hajautettuun ratkaisuun siirtyminen lisää nettiliikenteen ja tarvittavien viestien määrää? Selittyvätkö testeissä todetut linjasta poikkeavat tulokset paluuliikenteen suurella määrällä kuten arveltiin?

## Lähdeluettelo

- Abo16      Aboukhadijeh F., et al. Webtorrent projektin lähdekoodi, <https://github.com/feross/webtorrent> , 2016
- ALR10      Aalto S., Lassila P., Raatikainen, N., Savolainen P., Tarkoma S., P2P Video-on-Demand: Steady State and Scalability. *Global Telecommunications Conference (GLOBECOM 2010)*, IEEE, 2010
- Alv16      Alvestrand H., Real Time Protocols for Browser-based Applications, Internet draft, <https://tools.ietf.org/html/draft-ietf-rtcweb-overview-15> , 2016
- BPD10      Bakker A., Petrocco R., Dale M., Gerber J., Grishchenko V., Rabaioli D., Pouwelse J., Online Video Using BitTorrent and HTML5 Applied to Wikipedia. *2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P)*, IEEE, 2010
- BTS16      BitTorrent-protokollan spesifikaatio, <https://wiki.theory.org/BitTorrentSpecification> , 2016
- Coh03      Cohen B., Incentives Build Robustness in BitTorrent. *In Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, 2003
- CSD07      Choe Y., Schuff D., Dyaberi J., Pai, V., Improving VoD server efficiency with Bittorrent. *Proceedings of the 15th ACM international conference on Multimedia*, ACM, 2007
- DJT96      Deshpande Y., Jenkins R., Taylor S., Use of simulation to test client-server models. *Proceedings of the 28th conference on Winter simulation*, ACM, 1996
- DKP10      Dyaberi J., Kannan K., Pai, V., Storage optimization for a peer-to-peer video-on-demand network. *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, ACM, 2010
- Dut13      Dutton S., WebRTC in the real world: STUN, TURN and signaling, <http://www.html5rocks.com/en/tutorials/webrtc/infrastructure/> , 2013
- EDB08      Ekler P., Devai, I., Bakos B., Kiss A.J., BitTorrent Based Solution for

- Efficient Content Sharing on Next Generation Networks. *Next Generation Internet Networks*, IEEE, 2008
- HBf14 Hickson I., Berjon R., Faulkner S., Leithead T., Doyle Navara E., O'Connor E., Pfeiffer S., HTML5, W3C Recommendation, <http://www.w3.org/TR/html5/> , 2014
- HSR07 Hietanen H., Savolainen P., Rimey K., Browser-Based Peer-to-Peer Clients and Copyright Infringement. *Third International Conference on Automated Production of Cross Media Content for Multi-Channel Distribution*, IEEE, 2007
- JaM04 Jawaheer G., McCann J., Building a self-adaptive content distribution network. *Proceedings of 15th International Workshop on Database and Expert Systems Applications*, IEEE, 2004
- Kri16 Krill P., Oracle hops on the bandwagon to dump Java browser plug-in, <http://www.infoworld.com/article/3026969/java/oracle-hops-on-the-bandwagon-to-dump-java-browser-plug-in.html> , 2016
- KSK08 Kaune S., Stolzenburg J., Kovacevic A., Steinmetz R., Cuevas R., Understanding BitTorrent's Suitability in Various Applications and Environments. *The Third International Multi-Conference on Computing in the Global Information Technology*, IEEE, 2008
- LoN08 Loewenstern A., Norberg A., DHT protokolla, [http://www.bittorrent.org/beps/bep\\_0005.html](http://www.bittorrent.org/beps/bep_0005.html) , 2008
- Ris14 Ristic D., WebRTC data channels, <http://www.html5rocks.com/en/tutorials/webrtc/datachannels/> , 2014
- RMM08 Rosenberg J., Mahy R., Matthews P., Wing D., Session Traversal Utilities for NAT (STUN), RFC 5389, <https://tools.ietf.org/html/rfc5389> , 2008
- Ros10 Rosenberg J., Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols, RFC 5245, <https://tools.ietf.org/html/rfc5245> , 2010
- ShK06 Shibsankar D., Kangasharju J., Evaluation of network impact of content distribution mechanisms. *Proceedings of the 1st international conference on*

*Scalable information systems*, ACM, 2006

- SRS02 Stavrou A., Rubenstein D., Sahu S., A lightweight, robust P2P system to handle flash crowds. *Proceedings of 10th IEEE International Conference on Network Protocols*, IEEE, 2002
- SRT08 Savolainen, P., Raatikainen, N., Tarkoma, S., Windowing BitTorrent for Video-on-Demand: Not All is Lost with Tit-for-Tat. *2008 IEEE Global Telecommunications Conference*, IEEE, 2008
- Web16 Webtorrent projektin demo- ja kotisivu, <https://webtorrent.io/> , 2016
- WSW16 Wolenetz M., Smith J., Watson M., Colwell A., Bateman A., Media Source Extensions, W3C Candidate Recommendation, <https://www.w3.org/TR/media-source/> , 2016
- You16 YouTube HTML5 Video Player, <https://www.youtube.com/html5> , 2016
- ZAC13 Zhao M., Aditya P., Chen A., Lin Y., Haeberlen A., Druschel P., Maggs B., Wishon B., Ponc M.. Peer-Assisted Content Distribution in Akamai NetSession. *Proceedings of the 2013 conference on Internet measurement conference*, ACM, 2013
- ZCB11 Zeilemaker N., Capotă M., Bakker A., Pouwelse J., Tribler: P2P media search and sharing. *Proceedings of the 19th ACM international conference on Multimedia*, ACM, 2011
- ZZF12 Zhou F., Zhang L., Franco E., Mislove A., Revis R., Sundaram R., WebCloud: Recruiting social network users to assist in content distribution. *11th IEEE International Symposium on Network Computing and Applications (NCA)*, IEEE, 2012
- ZZM13 Zhang L., Zhou F., Mislove A., Sundaram R., Maygh: building a CDN from client web browsers. *Proceedings of the 8th ACM European Conference on Computer Systems*, ACM, 2013

## Liite 1. Koodiesimerkki

Koodiesimerkki on JavaScript node.js koodia ja se kuvaa palvelinpään käynnistämistä. Käänteinen välipalvelin, HTTP palvelin ja Bittorrent seurantapalvelin käynnistetään. Jokainen palvelimelle saapuva pyyntö johdetaan eteenpäin oikealle käsittelijälle käänteisen välipalvelimen toimesta. Seurantapalvelimelle saapuvat upgrade pyynnöt käsitellään ja HTTP protokollasta ”päivitetään” (upgrade) websocket protokollaan.

```
/**
 * HTTP reverse proxy server.
 */

var auto = require('run-auto')
var config = require('../config')
var cp = require('child_process')
var debug = require('debug')('webtorrent-ww:router')
var downgrade = require('downgrade')
var fs = require('fs')
var http = require('http')
var httpProxy = require('http-proxy')
var path = require('path')
var unlimited = require('unlimited')

unlimited()

var proxy = httpProxy.createProxyServer({
  xfwd: true
})

function onRequest (req, res) {
  if (req.headers.host === '192.168.1.10' ||
    req.headers.host === '192.168.1.10:' + config.ports.router.https) {
    proxy.web(req, res, { target: 'http://127.0.0.1:' + config.ports.tracker.http })
  } else {
    proxy.web(req, res, { target: 'http://127.0.0.1:' + config.ports.web })
  }
}

function onUpgrade (req, socket, head) {
  proxy.ws(req, socket, head, { target: 'ws://127.0.0.1:' + config.ports.tracker.http })
}

var httpServer = http.createServer(onRequest)
httpServer.on('upgrade', onUpgrade)

var httpServerT = http.createServer(onRequest)

var web, tracker
```

```

auto({
  httpServer: function (cb) {
    httpServer.listen(config.ports.router.http, config.host, cb)
  },
  httpServerT: function (cb) {
    httpServerT.listen(config.ports.router.https, config.host, cb)
  },
  tracker: function (cb) {
    tracker = spawn(__dirname + '/tracker')
    tracker.on('message', cb.bind(null, null))
  },
  web: function (cb) {
    web = spawn(__dirname + '/web')
    web.on('message', cb.bind(null, null))
  },
  downgrade: ['httpServer', 'httpServerT', 'tracker', function (cb) {
    downgrade()
    cb(null)
  }]
}, function (err) {
  debug('listening on %s', JSON.stringify(config.ports.router))
  if (err) {
    console.log(err)
    throw err
  }
})

function onError (err) {
  console.error(err.stack || err.message || err)
}

function spawn (program) {
  var child = cp.spawn('node', [ program ], {
    stdio: [ process.stdin, process.stdout, process.stderr, 'ipc' ]
  })
  child.on('error', onError)
  return child
}

process.on('uncaughtException', function (err) {
  onError(err)

  // kill all processes in the "process group", i.e. this process and the children
  try {
    process.kill(-process.pid)
  } catch (err) {
    console.log(err)
  }
})

```