

Aaro Altonen

# AVOIMEN LÄHDEKODIN HEVC-PIL- VITRANSKOODAUSJÄRJESTELMÄ

Informaatioteknologian ja viestinnän tiedekunta  
Kandidaatin työ  
Elokuu 2019

# TIIVISTELMÄ

Aaro Altonen: Avoimen lähdekoodin HEVC-pilvitranskoodausjärjestelmä  
Kandidaatin työ  
Tampereen yliopisto  
Tietoliikennetekniikka  
Elokuu 2019

---

Kyky tallentaa valtavia määriä videokuvaa vaatii helppokäyttöisiä ja tehokkaita videonkoodausjärjestelmiä, joiden avulla voidaan mukautua rajallisiin lähetys- ja tallennuskapasiteetteihin. Tässä työssä esitellään avoimen lähdekoodin pilvipalvelu, jonka avulla pystyy transkoodaamaan videoita H.265/HEVC-formaattiin. Vaihtoehtoisia kaupallisia toteutuksia on saatavilla, mutta ne ovat maksumuurien takana. Komentorivikäyttöliittymien käyttäminen taas vaatii syvällistä ymmärtämistä pakkausprosessista parhaan mahdollisen laadun ja nopeuden saavuttamiseksi. Esitelty järjestelmä on helppokäyttöinen, avoimen lähdekoodin toteutus, joka tekee siitä helposti lähestyttävän myös ei-teknisesti valveutuneille käyttäjille. Se on rakennettu käyttämällä FFmpeg-multimediatyökalua, jonka avulla pystyy dekodeamaan paljon erilaisia sisääntuloja sekä Kvazaar-nimistä HEVC-koodainta videonpakkaukseen.

Avainsanat: High Efficiency Video Coding (HEVC), Kvazaar HEVC-koodain, FFmpeg, Software as a Service (SaaS), pilvienkoodaus/-transkoodaus

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

# ALKUSANAT

Haluan kiittää Ultra Video Groupin työkavereitani, joilta sain paljon hyviä ehdotuksia ja apua ongelmatilanteissa. Erityiskiitokset myös Joni Räsäselle, joka auttoi järjestelmän testauksen ja käyttöliittymäsuunnittelun kanssa, sekä Marko Viitaselle, jolta sain ohjausta teknisissä haasteissa ja apua järjestelmän implementoinnissa.

Tampereella, 28.8.2019

Aaro Altonen

# SISÄLLYSLUETTELO

1. JOHDANTO.....	1
2. VIDEONPAKKAUS .....	2
2.1 Pakkaamaton raakakuva.....	2
2.2 Häviöllinen ja häviötön pakkaus.....	2
2.3 HEVC-videonpakkausstandardi .....	3
2.4 Kvazaar-videonpakkausohjelmisto .....	3
3. TRANSKOODAUS .....	4
3.1 Yleistä transkoodauksesta .....	4
3.2 Pilvitranskoodaus .....	5
3.3 Aiemmat pilvitranskoodausratkaisut .....	6
4. TOTEUTETTU PILVITRANSKOODAUSJÄRJESTELMÄ.....	7
4.1 Tekniset yksityiskohdat .....	7
4.2 Järjestelmän arkkitehtuuri .....	7
4.3 Käyttöliittymä .....	8
4.4 Taustajärjestelmä .....	10
4.4.1 Socket .....	10
4.4.2 Server .....	11
4.4.3 Parameter Manager .....	11
4.4.4 Worker.....	12
4.5 Järjestelmän toiminta .....	12
4.5.1 Videon lähetys ja validointi .....	13
4.5.2 Videon enkoodaus/transkoodaus .....	13
4.5.3 Videon muxaus .....	13
4.5.4 Videon tallennus.....	14
5. TOTEUTETUN JÄRJESTELMÄN SUORITUSKYKY JA VERTAILU.....	15
5.1 Suorituskyky .....	15
5.2 Vertailu aiempiin toteutuksiin .....	16
6. YHTEENVETO .....	17
LÄHTEET .....	18

## LYHENTEET JA MERKINNÄT

HEVC	High Efficiency Video Coding, pakkausstandrdi
FFmpeg	Multimediatyökalu videon- ja äänenkäsittelyä varten
AVC	Advanced Video Coding, videonpakkausstandardi
AV1	AOMedia Video 1, videonpakkausstandardi
RGB	Raakakuvaformaatti
YUV	Raakakuvaformaatti
YUV 4:2:0	Kvazaarin tukema raakakuvaformaatti
FPS	Frames per second, kuvataajuus
NAL	Network Abstraction Layer
RTP	Real-time Transport Protocol
CABAC	Context-adaptive binary arithmetic coding
TCP	Transmission Control Protocol

# 1. JOHDANTO

High Efficiency Video Coding (HEVC/H.265) [1] on vuonna 2013 julkaistu videonpakkausstandardi. Sen tarkoitus on vähentää bittinopeutta jopa 50% edelliseen MPEG AVC/H.264 -standardiin verrattuna [2]. Tiedostokoon pienentäminen kuulostaa hyvältä kenelle hyvänsä, joka tallentaa, prosessoi tai siirtää verkossa suuria määriä videokuvaa. Itse purkuun ja pakkaukseen käytetyt työkalut ovat kuitenkin monimutkaisia käyttää ja vaativat paljon pohjatietoa. Manuaalisesti videokuvan muuntaminen HEVC-formaattiin on hankalaa. Täten nämä työkalut ja tilallisesti pienikokoisempi videokuva ovat usein ei-teknisten käyttäjien ulottumattomissa.

Ongelman ratkaisuksi on kehitetty helppokäyttöisiä pilvitranskoodauspalveluita, joiden kautta video voidaan uudelleenpakata haluttuun formaattiin. Kehittyneimmät palvelut tarjoavat tuen suurelle määrälle erityyppisiä sisääntuloja ja ulostuloja (esim. HEVC, AVC, ja AV1). Näiden palvelujen avulla käyttäjä voi muuttaa haluamiensa videoiden formaattia esimerkiksi AVC:stä HEVC:ksi tallennustilaa säästääkseen.

Monet näistä palveluista ovat kuitenkin maksumuurien takana ja tarjoavat palveluitaan minuuttihinnoittelulla. Tarjolla on myös jonkin verran akateemisia ja avoimia toteutuksia, mutta nämä eivät ole enää ylläpidettyjä ja tarjoavat vain hyvin rajoitetun määrän sisään-/ulostuloja tai eivät ole ollenkaan kuluttajien saatavilla. Tässä työssä esitelty järjestelmä on avointa lähdekoodia sekä tarjoaa suuren määrän niin sisään- kuin ulostulojakin ja pakkaa videot käyttäen HEVC-koodekkia.

Luvussa kaksi käydään läpi yleistä teoriaa videonpakkauksesta ja HEVC:stä. Luku kolme selvittää mitä transkoodauksella tarkoitetaan ja esittelee sen alakategorian: pilvitranskoodauksen. Luku neljä vertailee tämänhetkisiä pilvitranskooderitoteutuksia. Luvussa viisi esitellään itse pilvitranskoodausjärjestelmä. Se käy läpi järjestelmän tekniset yksityiskohdat, selvittää miltä järjestelmän arkkitehtuuri näyttää ja miten koko prosessi toimii alusta loppuun. Luku kuusi esittää yhteenvedon tutkimuksesta.

## 2. VIDEONPAKKAUS

Videonpakkaus on tekniikka, jossa videokuvaa tiivistetään sen alkuperäisestä koosta pienempään. Se on tarpeellista esimerkiksi tilansäästöllisistä syistä tai jos halutaan lähettää videokuvaa Internetin välityksellä. Raakakuva vie suhteessa hyödyllisen informaation määrään nähden liikaa kaistaa/levytilaa. Sen takia on kehitetty tekniikoita pakata tämä informaatio pienempään kokoon.

### 2.1 Pakkaamaton raakakuva

Raakakuva on formaatti, jossa videota ei ole käsitelty millään tapaa. Raakavideota saadaan esimerkiksi suoraan videokameroilta. Raakavideossa tallennetaan kaikki saatu tieto. Esimerkiksi Full HD -video 1920x1080-resoluutiolla ja 30 FPS -kuvataajuudella vie minuutissa  $1920 * 1080 * 30 * 60 = 3\,732\,480\,000$  tavua eli 3.7 gigatavua.

Raakakuvaformaateista yleinen on RGB, jossa punaista, vihreää ja sinistä yhdistellen luodaan kuva. Tämä on kuitenkin tilallisesti huono tapa tallentaa kuvaa, sillä ihmissilmä ei ole hyvä erotamaan värieroja. Täten toinen suosiota saavuttanut formaatti on YUV, jossa ihmisen kyky havaita värejä on otettu huomioon. Siinä Y-signaali määrittää luminanssia eli kirkkautta ja U- ja V-signaalit määrittävät väriä. Y-signaalia on suhteessa U- ja V-signaaleihin paljon enemmän. [3] Mustavalkotelevisiolähetykset sisälsivät alun perin vain Y-signaalin, mutta kun haluttiin lähettää saman lähetyksen yhteydessä myös RGB-väriä, lisättiin väri signaalin U- ja V-komponentteihin. Mustavalkotelevisiot käyttivät ainoastaan signaalin Y-komponentin kuvan piirtoon, mutta väritelevisiot huomioivat myös U- ja V-komponentit. [4]

YUV 4:2:0 on siitä poikkeava formaatti, että Y-, U- ja V-komponentit ovat järjestetty niin että ensin on pelkästään Y-komponenttia, sitten U-komponenttia, ja lopulta V-komponenttia. [3]. Esitetyn pilvitranskooderin käyttämä Kvazaar-koodain hyväksyy sisääntulokseen ainoastaan YUV 4:2:0 -formaattissa olevaa raakakuvaa.

### 2.2 Häviöllinen ja häviötön pakkaus

Tarve videonpakkaukselle on ilmeinen, kun näkee paljonko tilaa raakakuvan tallentaminen vie. Videonpakkauksessa on käytössä kaksi selkojakoista kategoriaa: häviötön ja häviöllinen.

Häviötön pakkaus on tekniikka, jossa pakkausoperaatio ei hävitä mitään alkuperäisestä tiedostosta, mutta tiivistää videota pienempään tekniikoilla, jotka eivät vaadi tiedon poistamista. Sitä käytetään esimerkiksi tekstin pakkaamiseen, jossa tietoa ei saa hävitä. [5] Klassinen esimerkki häviöttömästä pakkauksesta on Huffmannin koodaus, jossa tekstissä useasti esiintyvät merkit pyritään esittämään lyhyemmällä esitysmuodoilla tilaa säästääkseen [6]. AVC/H.264 sekä HEVC/H.265 käyttävät häviöttömään pakkaukseen *Context-adaptive binary arithmetic coding* (CABAC) -nimistä tekniikkaa, joka on videonpakkaukseen erikoistunut aritmeettisen koodauksen malli.

Häviöllinen pakkaus taas on tekniikka, jossa pakkauksen yhteydessä poistetaan redundanttia tietoa. Sellaiseksi tiedoksi voidaan laskea pakkausmenetelmästä riippuen esimerkiksi paikoillaan pysyvät objektit videokuvassa tai tieto, jota ihmissilmä tai -korva ei pysty havaitsemaan. [7] Häviöllinen pakkaus kykenee pakkaamaan häviötöntä paljon tehokkaammin, koska sen sallitaan poistaa informaatiota videokuvasta. Haaste pakattua videokuvaa häviöllisesti transkoodatessa on artifaktien [8] eli vääristymien moninkertaistuminen, ja usean transkoodauksen jälkeen video voi olla katselukelvoton. Esimerkki häviöllisestä pakkauksesta voisi olla raakakuvaa käsittelevä chroma subsampling -tekniikka, jossa esimerkiksi RGB32-formaatissa oleva raakakuva muutetaan YUV 4:2:0 -formaattiin [3]. Tämä pakkaustekniikka poistaa pakattavasta signaalista väriin liittyviä yksityiskohtia, sillä ihmissilmä on huono havaitsemaan tarkkoja värieroja. Yksityiskohtien poistaminen tarkoittaa kuitenkin sitä, että alkuperäistä videota ei pystytä enää pakatusta tiedostosta palauttamaan.

AVC/H.264 on hyvin laaja-alaisesti adaptoitu häviöllinen videonpakkausstandardi, jota käyttää muun muassa Youtube [9] ja Netflix [10]. AVC/H.264-videonpakkausstandardin suurimmat myyntivalitit sitä julkaistaessa olivat jopa 50% pienempi bittinopeus verrattuna edellisiin koodekkeihin sekä NAL-tekniikka, jonka avulla videokuva oli mielekkäästi jaettavissa pienempiin palasiin ja ne voitiin lähettää Internetin yli esimerkiksi RTP-paketeissa [11].

## 2.3 HEVC-videonpakkausstandardi

High Efficiency Video Coding (HEVC/H.265) [1] on vuonna 2013 julkaistu videonpakkausstandardi. Sen tarkoitus on vähentää bittinopeutta jopa 50% edelliseen MPEG AVC/H.264 -standardiin verrattuna [2]. HEVC on implementoitu ratkaisemaan kaksi keskeistä haastetta: videokoon kasvamisen ja rinnakkaislaskennan tarpeen videonpakkauksessa [1]. Tämä siksi, että AVC ei kykene pakkamaan omaa teoreettista maksimiaan paremmin videota, vaikka tarve pienempikoiselle videolle kasvaa koko ajan.

HEVC:n merkittävimmät ominaisuuden videonpakkauksessa ovat:

- Coding Tree Unit (CTU), jonka avulla video pystytään pakkaamaan tehokkaammin
- Parempi tuki pakkausprosessin rinnakaistamiselle
- Wavefront Parallel Processing (WPP), joka mahdollistaa paremman pakkaustehokkuuden

Lyhykäisyydessään HEVC-yhteensopiva pakkaus toimii näin: sisään tuleva kuva jaetaan lohkoihin, ja nämä lohkokoot tallennetaan videon yhteyteen, jotta video pystytään myös purkamaan. Videosekvenssin ensimmäinen ja joka N:nnäs kuva käyttäjän määrittämisestä riippuen pakataan intrapicture prediction -tekniikalla, jossa pakattavalla kuvalla ei ole riippuvuuksia muihin kuviin. Näitä kuvia kutsutaan Intra-kuviksi, ja niitä käytetään referenssikuvina, kun videota puretaan. Intrakuvien pakkaaminen vie kaikkein eniten aikaa ja ne ovat myös tilallisesti suurimpia. Muuten käytetään interpicture prediction -tekniikkaa, jossa Inter-kuva käyttää edellisiä sekä seuraavia kuvia kuvan pakkaukseen. Inter-kuvat vievät Intra-kuvia paljon vähemmän tilaa. [1]

## 2.4 Kvazaar-videonpakkausohjelmisto

Kvazaar on palkittu avoimen lähdekoodin HEVC-koodain [12], [13]. Sitä kehittää Tampereen Yliopistolla Ultra Video Group -niminen tutkimusryhmä ja se on saatavilla GNU Lesser General Public License -lisenssin alaisena. Kvazaar mahdollistaa reaaliaikaisen 4K-videon pakkaamisen ultrafast-asetuksella ja saavuttaa veryslow-asetuksella miltei saman koodaustehokkuuden kuin HEVC-referenssi-implementaatio [14]. Kvazaar tukee lähes kaikkia HEVC-koodaintyökaluja kuten

- Wavefront Parallel Processing (WPP)
- Overlapped Wavefront
- Multithreading
- Deblock filter
- Sample Adaptive Offset (SAO)
- Sub-pixel motion estimation
- Prediction unit depth limitation
- Bi-prediction
- Rate control

Laskennallisesti vaativimmat funktiot on optimoitu käyttämällä SSE4.1- ja AVX2-käskeyntäalajennoksia x86/x64-arkkitehtuurille. Suurinta osaa koodaintyökaluista hallitaan komentorivityökalun avulla. Kvazaarista on saatavilla myös kirjasto, joten sitä on mahdollista käyttää myös ohjelmointirajapinnan (API) kautta. Kvazaar tukee raakakuvaformaattia YUV 4:2:0 sisääntulona ja ulostuloina se tukee Main- ja Main 10 -standardiprofileja.

## 3. TRANSKOODAUS

Transkoodaus, ja tarkemmin videotranskoodaus, on tekniikka, jossa formaatissa A oleva videosisääntulo muunnetaan videoformaattiin B. Videoformaatti määritellään esimerkiksi bittinopeuden, kuvataajuuden, tai tilallisen resoluution mukaan. [15] Video voidaan joutua transkoodaamaan, jos huomataan, että kohdejärjestelmä ei tue esimerkiksi videon tämänhetkistä kuvataajuutta tai koodekkia.

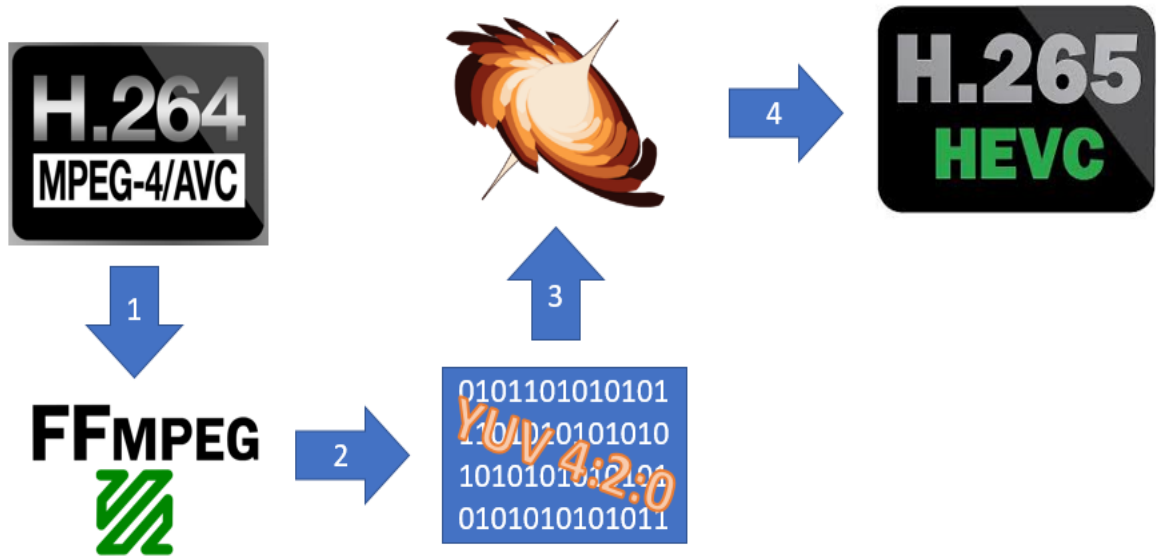
### 3.1 Yleistä transkoodauksesta

Transkoodausen tarve ilmeni jo televisioiden tullessa markkinoille, sillä monet ohjelmat oltiin kuvattu studiolaadulla (korkea bittinopeus), mutta näin korkealaatuisen signaalin siirtäminen ei ole taloudellisesti kannattavaa tai edes järkevää, mikäli vastaanottava laite ei pysty signaalia purkamaan. Transkoodaus mahdollisti tämän korkealaatuisen signaalin muuttamisen sellaiseen muotoon, että se pystyttiin kuljettamaan kuluttajien kotiin. [15] Nykyään transkoodausta käytetään paljon erilaisissa "Video on Demand" -palveluissa, joissa jokin videotiedosto pitää pystyä toistamaan hyvin erilaisilta laitteilta (esim. älypuhelimelta tai televisiosta).

Tarve erilaisille bittinopeuksille ja kuvataajuuksille pystyttäisiin täyttämään myös skaalattavilla pakkaustekniikoilla, mutta niissä ongelmaksi ilmenee sekä enkooderien että dekodeerien monimutkaisuus, mutta myös adaptiivisuuden puute. Video joko pakataan alhaisella bittinopeudella, joka johtaa huonoon kuvanlaatuun tai korkealla bittinopeudella, joka taas voi johtaa siihen, että video ei pääse liikkumaan verkossa sulavasti. [15]. Transkoodaus on saavuttanut täten suurempaa suosiota mukautuvuutensa ansiosta.

Transkoodaus voidaan jakaa karkeasti kahteen luokkaan: homogeeniseen ja heterogeeniseen. Homogeenisessä transkoodauksessa videon formaatti ei muutu, mutta sen bittinopeutta saataan nostaa/laskea tai siitä voidaan rajata esimerkiksi kiinnostava alue (engl. region of interest) ja pakata vain tämä alue uudelleen. Heterogeenisessä transkoodauksessa videon formaatti muuttuu (esim. AVC -> HEVC -muunnos). Se saattaa sisältää myös logon tai vesileiman lisäämisen transkoodattavaan videoon. [16]

Transkoodaus voidaan suorittaa suoralla digitaali-digitaali-konversiolla tai vaihtoehtoisesti purkamalla sisääntulo ensin johonkin raakaformaattiin ja pakkaamalla tämä raakakuva sitten uudelleen haluttuun formaattiin. Jälkimmäinen on saavuttanut suosiota, koska se on kahdesta esitellystä suosituimpi, sillä se on modulaarisempi sekä tukee suurempaa määrää eri formaatteja ja niiden välisiä konversioita. [16]. Kappaleessa 5 esitelty järjestelmä käyttää jälkimmäisen mainittua tekniikkaa juuri sen modulaarisuuden ansiosta.



**Kuva 1: AVC to HEVC -transkoodaus**

Kuva 1 esittää miten transkoodaus tapahtuu. Kuvan esimerkissä AVC/H.264-video ensin puretaan raakakuvaformaattiin (tässä tapauksessa YUV 4:2:0) FFmpeg-multimediatyökalulla. Raakavideo on esitetty laatikkona nolliä ja ykkösiä. Purkaminen eli dekoodaaminen tuottaa monta kertaa suuremman tiedoston kuin mikä FFmpeg:lle annettiin. Kun raakakuva on purettu, annetaan se Kvazaarille, joka pakkaa eli enkoodaa sen HEVC/H.265-formaattiin. Näin saatiin suoritettua "AVC -> HEVC" -transkoodaus.

### 3.2 Pilvitranskoodaus

Mobiililaitteiden määrän valtava räjähdys vaatii videon suoratoistopalveluilta mukautumista vaihteleville kaistanleveyksille ja ruutujen dimensioille. Näissä laitteissa ei kuitenkaan useasti ole varattu tehoja tai akkukapasiteettia videon transkoodaukselle, ja väärässä formaatissa olevan videon toimittaminen päätelaiteelle ja sen transkoodaaminen siellä aiheuttaisi turhaa ja tilanteesta riippuen suurtakin latenssia videonkatselussa.

Ongelmaan on kaksi ratkaisua: säilytää palvelimella kaikkia mahdollisia formaatteja tai transkoodaa pyynnön saapuessa video oikeaa formaattiin. Ensimmäisen ratkaisun ainoa hyvä puoli on se, että haluttu video mitä todennäköisimmin löytyy jo palvelimelta. Se ei ole kuitenkaan kustannustehokas ja tulee vaatimaan valtavia määriä tallennustilaa. Parempi ratkaisu on transkoodata video pyydettyyn formaattiin. Tällä tavoin ei tarvitse säilyttää kuin yksi kopio videotiedostosta. Ongelmaksi voi kuitenkin ilmetä ruuhka ja sen aiheuttamat latenssit. Jos useaa videotiedostoa transkoodataan samanaikaisesti, hidastuu kaikkien transkoodausprosessi, koska kaikki kilpailevat järjestelmän resursseista. Tähänkin on kuitenkin kehitetty ratkaisu [17].

Pilvessä tapahtuva transkoodaus mahdollistaa myös paljon enemmän käyttäjän tarpeisiin mukautuvia "Video on Demand" -palveluita. Esimerkiksi Youtubea tai Twitch.tv:tä on mahdollista käyttää hyvin erilaisilta laitteilta, ja tämä laitteiden monimuotoisuus tarjoaa haasteen palveluntarjoajille. Paras ratkaisu tähän on transkoodata video käyttäjän tarpeiden mukaan.

Tässä työssä esitelty pilvitranskooderi suorittaa hyvin yksinkertaista laadunvarmistusta (engl. Quality-of-Service) rajoittamalla samanaikaiset transkoodausprosessit viiteen. Tällä määrällä jonoa saa purettua nopeammin, mutta viisi rinnakkaista prosessia ei kuitenkaan jäädytä järjestelmää täysin.

### 3.3 Aiemmat pilvitranskoodausratkaisut

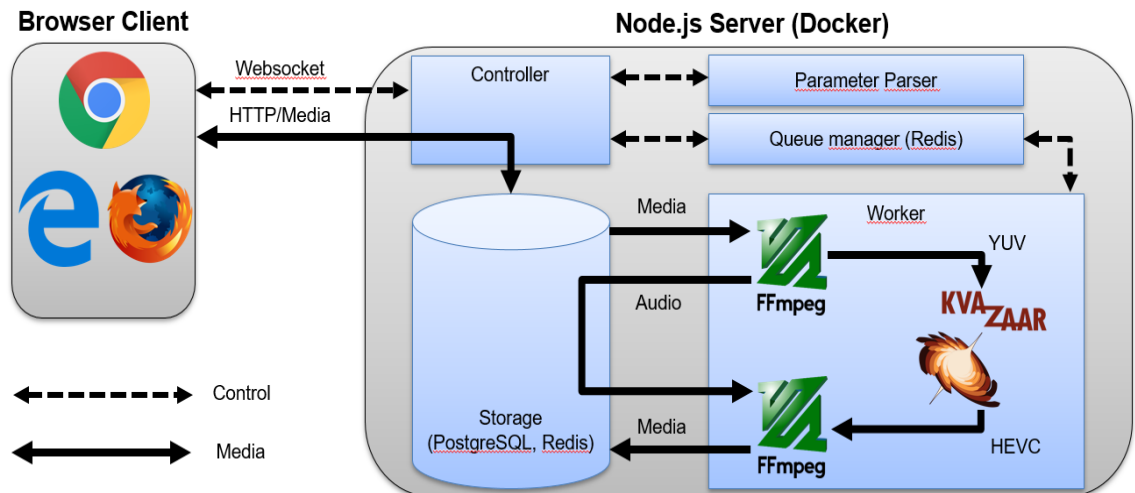
Kuten on jo aiemmin todettu, valmiita toteutuksia löytyy paljon. Kappaleessa 5.2 vertaillaan tarkemmin eri pilvitranskooderien ominaisuuksia. Kaupallisista mainitsemisen arvoisia ovat Amazon [18], Coconut [19], Qencode [20], ja Zencoder [21]. Nämä palvelut tarjoavat enkoodaus-/transkoodauspalveluja minuutti-/tavumääräisellä hinnoittelulla.

Myös akateemisia toteutuksia löytyy. Z. H. Chang ja muut [22] implementoivat reaaliaikaisen hajautetun järjestelmän suoratoistetun videon transkoodaamiseen. M. Chen ja muut [23] tutkivat rinnakkaista transkoodaamista, ja Y. Dong ja muut [24] esittelivät kontitettua (engl. containerized) videotranskoodausjärjestelmän helppokäyttöisyyttä ajatellen.

Myös joitakin avoimen lähdekoodin toteutuksia on saatavilla kuten Cloud Transcode [25], Morph [26], ja Snickers [27], joista kahta viimeksi mainittua ei enää ylläpidetä aktiivisesti. Cloud Transcode ja Snickers mahdollistavat tiedostonlatauksen vain HTTP-linkin tai Amazon S3:n kautta. Molemmat niistä ovat myös sidottu Amazonin pilvipalveluun nimeltä S3 ja niitä voi käyttää vain tarjottujen ohjelmointirajapintojen (API) kautta. Morph on rajoittunut ainoastaan MP4-sisääntulon ja AVC-ulostuloon tukemiseen. Tämän lisäksi näistä kaikista puuttuu käyttäjäystävällinen käyttöliittymä.

## 4. TOTEUTETTU PILVITRANSKOODAUSJÄR- JESTELMÄ

Esitely järjestelmä on Web-käyttöliittymän kautta käytettävä, avointa ja vapaata lähdekoodia (BSD 2-Clause -lisenssi) ja vapaasti käytettävissä osoitteessa: <http://ultravideo.cs.tut.fi/cloud>.



**Kuva 2: Järjestelmän korkean tason kuvaus**

Kuva 2 esittää korkealla tasolla, miten se toimii. Järjestelmän käyttö vaatii HTML5-yhteensopivan selaimen. Käyttäjä voi myös halutessaan ladata projektin Github-sivulta [28] Docker-tiedoston ja ajaa järjestelmää lokaalisti omalla koneella niin halutessaan.

### 4.1 Tekniset yksityiskohdat

Kvazaar Cloud Encoder on kirjoitettu enimmäkseen JavaScriptillä. Käyttöliittymä (engl. frontend) on implementoitu normaalin JavaScriptin lisäksi jQueryllä [29]. Ulkoasuun on käytetty Bootstrap-nimistä [30] ohjelmistokehystä (engl. software framework). Taustajärjestelmä (engl. backend) on kirjoitettu kokonaan Node.js:llä [31]. Järjestelmä käyttää hyväkseen useita avoimen lähdekoodin ohjelmia kuten Kvazaaria, FFmpeg:ä [32], Redis- ja PostgreSQL-tietokantoja [33], [34], sekä Docker-konttiyökalua [35].

### 4.2 Järjestelmän arkkitehtuuri

Järjestelmä on jaettavissa kahteen selkeään eri komponenttiin: käyttöliittymään ja taustajärjestelmään. Nämä keskustelevat keskenään Websockettien [36] avulla. Tällä tavoin saadaan interaktiivisesti päivittyvä käyttöliittymä, jota on paljon mielekkäämpi käyttää verrattuna traditionaaliseen sivulatausmekanismiin.

Websocket on suhteellisen moderni tekniikka, joka mahdollistaa full-duplex-yhteyden luonnin asiakkaan ja palvelimen välillä. Tällä tavoin voidaan tarjota low-overhead-kommunikointia päätepisteiden välillä. Sen avulla on myös yksinkertaista toteuttaa interaktiivisia kyselyitä palvelimelle. Websocket on toteutettu TCP:llä. [37]

### 4.3 Käyttöliittymä

Järjestelmän käyttöliittymä on kirjoitettu JavaScriptillä sekä jQuery- ja Bootstrap-nimisillä ohjelmistokehyksillä. Sivun avautuessa taustalla luodaan WebSocket, joka ottaa yhteyden taustajärjestelmään. Tämän WebSocketin kautta tapahtuu kaikki kommunikointi taustajärjestelmän kanssa (mm. parametrien tarkistus, tiedoston latauspyyntö). WebSocketin luonnin yhteydessä käyttäjälle luodaan uniikki käyttäjätunniste, jonka avulla taustajärjestelmä hallinnoi kaikkia siihen yhteydessä olevia käyttäjiä. Tämä käyttäjätunniste tallennetaan selaimen evästeisiin, mikäli käyttäjä on sen sallinut. Tämä mahdollistaa sen, että kun käyttäjä on ladannut tiedoston palvelimelle voi hän sulkea selaimen ja tulla myöhemmin lataamaan valmiin tiedoston. Järjestelmän käyttö ei kuitenkaan vaadi evästeiden käyttöä, mutta tällöin käyttäjä joutuu pitämään selaimen ikkunan avoimena, ettei WebSocket-yhteys tuhoudu.

Koska järjestelmä sallii suurien tiedostojen lataamisen (50 GB raakavideolle, 30 minuuttia pakattua videokuvaa), täytyy tiedoston latauksesta palvelimelle tehdä luotettavaa. Järjestelmä käyttää Resumable.js-nimistä [38] kirjastoa tiedostonlataukseen. Resumable.js on HTML5-ohjelmointirajapintaa [39] käyttävä JavaScript-kirjasto, joka pilkkoo tiedostot käyttäjän päässä pieniksi palasiksi ja lähettää nämä palaset taustajärjestelmälle mahdollistaen suurten tiedostojen luotettavan latauksen.



#### Kvazaar HEVC Cloud Encoder/Transcoder

Home My videos (0) About

Cloud accepts raw video in [several different formats](#), raw H.264 video and video in any container FFmpeg is able to decode and it encodes the input video to HEVC

Size limits: 30 minutes for containerized video, 50 GB for raw video

Drop video file here to upload or [select from your computer](#)

Selected encoding level: Ultrafast (fastest, lowest quality)

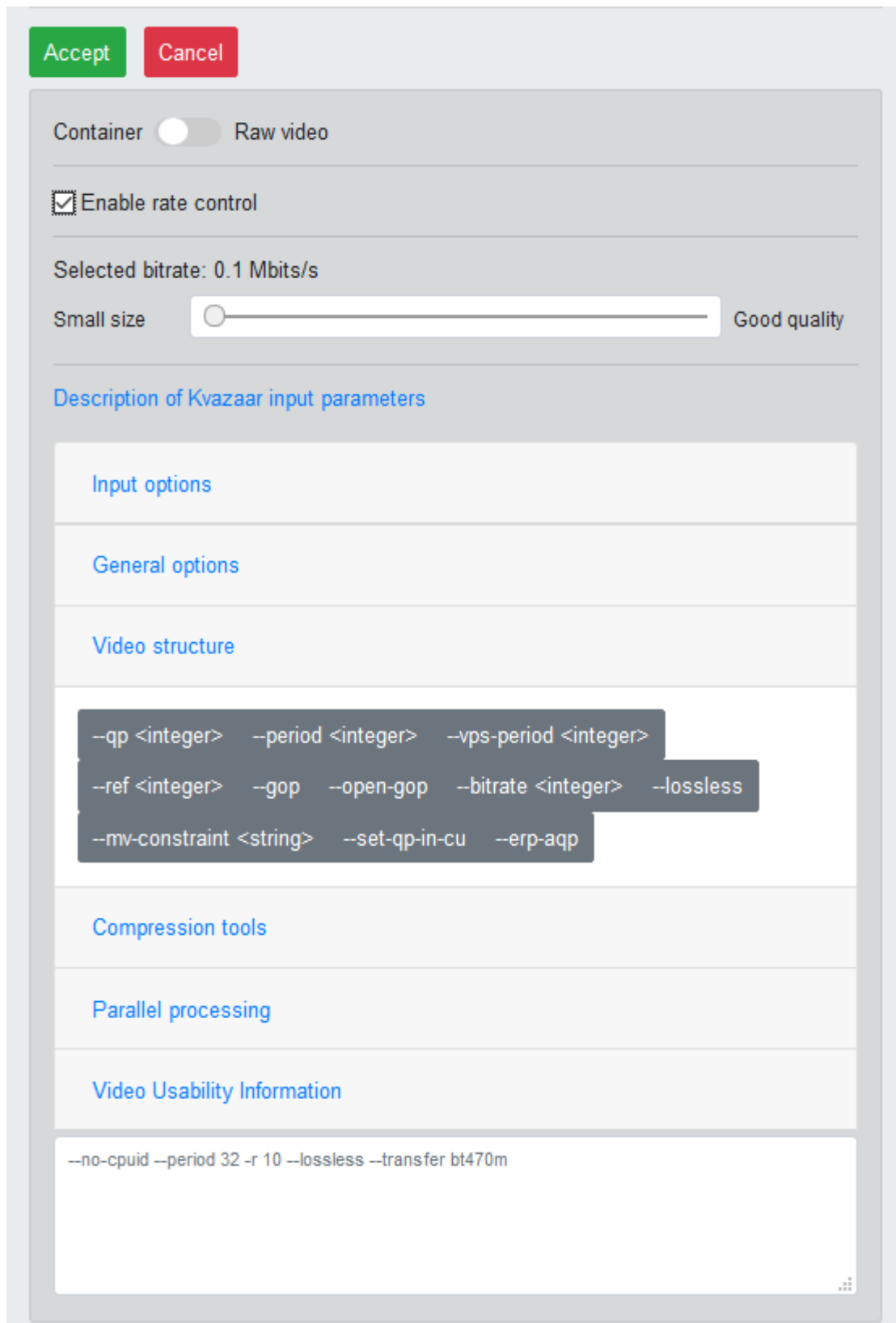
Good quality  High speed

[Advanced settings](#)

Output container

MP4

**Kuva 3: Järjestelmän pääsivu**



**Kuva 4: Lisäasetukset-näkymä**

Kuva 3 esittää, miltä järjestelmä päänäkymä näyttää. Se sisältää peruskäytön kannalta oleelliset asiat eli tiedoston, pakkaustyylin ja ulostuloformaatin valitsemisen. Käyttäjä pystyy liukusäätimen (engl. slider) avulla määrittämään haluamansa balanssin pakkausnopeuden ja -laadun välillä. Hitaampi pakkausnopeus tuottaa suurempaa bittinopeutta ja täten parempilaatuista

kuvaa. Nopea pakkausnopeus taas pyrkii pakkaamaan videon mahdollisimman nopeasti kuvan laadun kustannuksella.

Kuva 4 esittää järjestelmän lisäasetukset. Järjestelmälle on mahdollista antaa raakavideokuvaa, mutta se vaatii esimerkiksi kuvataajuuden ja resoluution määrittämisen. Käyttäjä voi halutessaan myös päättää minkälaisella bittinopeudella video tulee pakata käyttämällä Bitrate-liukusäädintä. Järjestelmä mahdollistaa myös pakkausprosessinustomoinnin parametrilasolla. Tämä tarkoittaa, että käyttäjä voi kirjoittaa kuvassa näkyvään laatikkoon Kvazaarille annettavia parametreja, joita sitten käytetään, kun videota aloitetaan pakkaamaan. Helppokäyttöisyyden vuoksi tuetut parametrit ovat myös lajiteltu ryhmittäin painikkeiksi, joita painamalla ne täydentyvät automaattisesti laatikkoon helpottaen prosessia.

## 4.4 Taustajärjestelmä

Taustajärjestelmä on täysin Node.js:llä kirjoitettu. Se jakautuu karkeasti neljään komponenttiin: Socket, Server, Parameter Manager ja Worker. Näistä jokainen käydään tarkemmin läpi alempana.

Taustajärjestelmän komponentit keskustelevat keskenään käyttämällä Redis-tietokantaa viestijonona ja lähettävät tietyn rakenteen omaavia viestejä toisilleen. Jos esimerkiksi Server haluaa lähettää viestin käyttäjälle tai yhdelle Workereista, lähettää se viestin ensin Socketille, joka edelleenlähettää tämän viestin sitten oikealle kohteelle.

Järjestelmä käynnistetään ajamalla Socket, joka luo viisi Worker-säiettä ja yhden säikeen Serverille. Tämän lisäksi käynnistetään Redis- ja PostgreSQL-tietokannat.

Järjestelmä tarjoaa muille komponenteille tietokantarajapinnan db.js-nimisen tiedoston kautta. Alun perin järjestelmä käytti SQLite-nimistä tietokantaa, mutta järjestelmän rinnakkainen toiminta aiheutti tietokantalukkoja, jotka jäädyttivät Worker-säikeiden toiminnan täysin. Tämän havainnon jälkeen siirryttiin käyttämään PostgreSQL-tietokantaa, joka käyttää row-level locking -tekniikkaa, jossa koko taulun sijaan vain muokattava tietue lukitaan. Tällä tavalla pystytään tarjoamaan viiden Worker-säikeen samanaikainen toiminta ja pääsy tietokantaan.

### 4.4.1 Socket

Socket on järjestelmän toiminannon kannalta hyvin tärkeä toimija. Se muodostaa ja ylläpitää yhteyttä käyttäjään, mutta myös tarjoaa järjestelmän muille komponenteille mahdollisuuden keskustella käyttäjän kanssa lähettämällä esimerkiksi reaaliaikaisia virheviestejä tai prosessiin liittyviä tilanpäivityksiä. Käynnistyessään Socket tarkistaa, että binäärit Kvazaarille ja FFmpeg:lle löytyvät järjestelmästä, sillä ilman niitä mitään ei pysty tekemään.

Socketin yksi tärkeimpiä tehtäviä on käyttäjien tekemien pyyntöjen validointi. Tilaa säästääkseen järjestelmä pyrkii käyttämään jo sinne ladattuja tiedostoja, mikäli se havaitsee, että käyttäjän lähettämä latauspyyntö koskee videota, joka palvelimelta jo löytyy. Latauspyynnön validoinnin ohessa tarkistetaan myös, että kaikki videota koskevat tiedot ovat valideja, ja näistä tarkistuksista vastuussa on Parameter Manager. Järjestelmään ei ole mahdollista luoda kahta samanlaista pyyntöä samalta käyttäjältä (sama tiedosto ja samat parametrit). Socket tarkistaa, että lähetetty pyyntö on uniikki ja mikäli se ei ole, ilmoitetaan käyttäjälle siitä ja hylätään lähetetty pyyntö.

Se on vastuussa myös Worker-toimijoiden "pingaamisesta". Socket siis lähettää tietyn intervallin välin Workereille viestin, joihin niiden täytyy vastata tietyn aikaikkunan sisällä. Mikäli näin ei tapahdu, katsotaan Workerin kuolleen ja sen tilalle luodaan uusi Worker, jotta järjestelmä voi jatkaa normaalia toimintaansa.

Järjestelmän komponenttien välinen kommunikaatio tapahtuu Request/Reply-pareilla. Käytännössä se tarkoittaa sitä, että esimerkiksi käyttäjän Websocket lähettää *uploadRequest*-viestin Socketille, joka validoi viestin yhteydessä annetut tiedot ja lähettää käyttäjälle *uploadReply*-viestin, jonka mukana kuljetetaan tieto siitä, voiko latauksen aloittaa.

Järjestelmässä on myös muutamia muita viestejä:

- *taskQuery*, jonka avulla käyttäjä saa listan tekemistään videolatauksista
- *cancelInfo*, jonka käyttäjä lähettää, kun hän keskeyttää käynnissä olevan prosessin
- sisäinen *cancelRequest*, jonka Socket lähettää Workerille, jos prosessi pitää jostain syystä pysäyttää

### Taulukko 1: Järjestelmän viestityypit

Viestin nimi	Vastaus vaadittu	Viestin selitys
downloadRequest	kyllä	Käyttäjä haluaa ladata videon, josta aiemmin tehnyt pyynnön
uploadRequest	kyllä	Käyttäjä haluaa tehdä uuden enkoodauspyynnön
deleteRequest	kyllä	Käyttäjä haluaa poistaa tekemänsä pyynnön järjestelmästä
cancelRequest	kyllä	Käyttäjä haluaa keskeyttää latauksen/enkoodauksen
optionsValidationRequest	kyllä	Validoi käyttäjän antamat Kvazaar-parametrit
pixelFormatValidationRequest	kyllä	Validoi käyttäjän antama pikseliformaatti
init	ei	Uusi käyttäjä yhdisti, luo sille uusi Clients-objekti
reinit	ei	Käyttäjä, joka on käyttänyt järjestelmää aiemminkin, yhdisti. Luo/päivitä sen tieto Clients-tilauksessa
taskQuery	kyllä	Käyttäjä klikkasi My Videos -linkkiä, hae kaikki hänen tekemät pyynnot tietokannasta
cancelInfo	ei	Käyttäjä keskeytti videon latauksen

## 4.4.2 Server

Server on järjestelmän toimija, joka on vastuussa tiedostonlataukseen liittyvistä asioista. Server sekä vastaanottaa käyttäjän lähettämät videonpalaset ja yhdistää ne, kun kaikki on vastaanotettu, valvoo tiedostolle annettuja rajoituksia ja keskeyttää latauksen, mikäli ehdot eivät täyty (liian iso tiedosto, ladattava tiedosto ei ole videotiedosto), ja siirtää ladatun tiedoston jonoon, kun lataus on valmis.

Käyttäjä lataa myös Serverin kautta valmistuneet tiedostot yhdistämällä osoitteeseen `/download/<tiedoston uniikki tunnistus>`. Myös käyttöliittymän tarvitsemat JavaScript-kirjastot ladataan Serverin kautta.

Server kommunikoi Socketin ja sen kautta käyttäjän kanssa Rediksen avustuksella. Se lähettää yllä määritettyjä *status*- ja *action*-viestejä Socketille, joka sitten lähettää ne tarpeen mukaan eteenpäin käyttäjälle. Näiden avulla keskeytetään esimerkiksi tiedostonlataus, jos huomataan, että se ei ole määritysten mukainen.

## 4.4.3 Parameter Manager

Parameter Manager on erilliseksi toimijaksi eriytetty järjestelmän sisäinen palvelu, jonka tehtävänä on validoida kaikki käyttäjältä saatu tieto. Tämä koskee sisään tulevan videon pikseliformaattia, resoluutiota, kuvataajuutta, resoluutiota sekä Kvazaarille annettavia parametreja. Suurin osa tiedoista pystytään validoimaan säännöllisillä lausekkeilla (engl. regular expression, regex). Kvazaarin parametrien kohdalla on kuitenkin jouduttu tekemään suunnittelupoikkeus ja implementoimaan erillinen tietorakenne ja sitä käsittelevät rutiinit, jotta parametrit voidaan validoida tehokkaasti.

Kvazaarin parametrit ovat tallennettu hajautustauluun (engl. hashmap), josta niitä pystyy etsimään nopeasti nimen perusteella. Jokaisen parametrin oheen tallennetaan tieto siitä, onko tämä parametri käyttäjän määritettävissä (ignored-lippu, totuusarvot true/false). Ignored-tiedon lisäksi tallennetaan tieto siitä, minkälaisia arvoja parametrilta voi olettaa. Tämä muuttuja on lista erityyppisiä arvoja (esimerkiksi Boolean-totuusarvoja, merkkijonoja tai säännöllisiä lausekkeita). Algoritmi 1 kuvaa pseudokoodin muodossa, miten parametrien validointi tapahtuu.

```

bool validoiParametri(Object arvo, Object[] validitArvot)
{
    for (int i = 0; i < validitArvot.size(); ++i) {
        if (typeof(validitArvot[i]) == string) {
            return arvo == validitArvot[i];
        }

        if (typeof(validitArvot[i]) == regex) {
            // tarkista matchaako annettu arvo oletettu regexiä
            return validitArvot[i].match(arvo);
        }

        if (typeof(validitArvot[i]) == boolean) {
            if (validitArvot[i] == arvo)
                return true;
        }
    }
}

```

**Algoritmi 1: Kvazaar-parametrien validointi**

#### 4.4.4 Worker

Worker on Socketin lisäksi hyvin merkityksellinen järjestelmän osa. Workerin vastuulla on videon purkaminen, mahdollinen transkoodaaminen oikeaan raakakuvaformaattiin, raakakuvan enkoodaaminen HEVC-formaattiin, sekä HEVC-videon siirtäminen alkuperäisen ääniraidan kanssa käyttäjän pyytämään säiliömuotoon (engl. container format).

Järjestelmän käynnistyessä Socket luo viisi Worker-säiettä, jotka ovat vastuussa käyttäjän pyyntöjen käsittelystä. Kaikki Workerit lukevat samaa pyyntöjonoa, jonne käyttäjien tallentamat pyynnöt tallennetaan. Pyyntöt käsitellään FIFO-menetelmällä.

Tämän lisäksi Worker kuuntelee myös viestejä Socketilta (mm. *ping*- ja *cancelInfo*-viestit) ja vastaa niihin asianmukaisesti. Sen on kyettävä keskeyttämään mikä tahansa käynnissä oleva prosessi, joten se pitää jatkuvasti kirjaa siitä mitä se on tekemässä ja on valmis lopettaa prosessin niin pyydettyäessä.

Worker lähettää prosessin edetessä siihen liittyviä reaaliaikaisia viestejä, jotka kertovat missä kohtaa prosessi on tällä hetkellä menossa (*Decoding*, *Encoding*, *Post-processing* jne.).

Worker tarjoaa myös reaaliaikaista tietoa pakkausprosessin etenemisestä, ja käyttäjä voi My Videos -näkyvästä nähdä kuinka suuri osa videosta prosentuaalisesti on pakattu.

### 4.5 Järjestelmän toiminta

Järjestelmän toiminnan voi kuvata nelivaiheisena prosessina: lataus ja validointi, enkoodaus/transkoodaus, muxaus ja lataus. Käyttäjä voi halutessaan pysäyttää prosessin milloin hyvänsä. Järjestelmä ei tallenna tietoja keskeytyneistä prosesseista eli katumapälle tullessaan joutuu käyttäjä aloittamaan prosessin ihan alusta.

### 4.5.1 Videon lähetys ja validointi

Prosessi alkaa, kun käyttäjä valitsee koneeltaan tiedoston, jonka haluaa transkoodata ja raahaa sen näkymän laatikkoon. Mikäli kyseessä on raakavideo, ja järjestelmä huomaa sen, pyrkii se automaattisesti täyttämään videotiedoston nimestä mahdollisesti löytyvät kuvatajuuden, resoluution ja bittisyvyyden. Mikäli näitä ei videotiedoston nimestä pystytä löytämään, joutuu käyttäjä manuaalisesti ne täyttämään.

Videon valinnan jälkeen voi käyttäjä muuttaa prosessiin liittyviä asetuksia kuten prosessin nopeutta ja ulostulon formaattia. Järjestelmän yksi suuri etu on se, että se tukee valtavaa määrää eri formaatteja ja käyttäjän ei tarvitse huolehtia siitä. Käyttäjä voi halutessaan muuttaa Advacend settings –nappia painamalla useita itse pakkausprosessiin liittyviä asetuksia kuten Kuva 4 osoittaa.

Kun käyttäjä on valmis aloittaan pakkausprosessin, klikkaa hän Encode-nappia, joka lähettää prosessiin liittyvät tiedot taustajärjestelmälle käsiteltäviksi *uploadRequest*-viestin sisällä. Taustajärjestelmä muun muassa tarkistaa, että kaikki tiedot ovat oikeassa formaatissa ja että mitään tarpeellista ei ole jätetty täyttämättä. Se tarkistaa myös löytyykö ladattava tiedosto jo palvelimelta, ja jos löytyy niin tiedostoa ei ladata sinne uudelleen. Taustajärjestelmä vastaa käyttäjälle *uploadReply*-viestissä, jossa se kertoo, hyväksyttiinkö latauspyyntö, pakkauspyyntö, molemmat vai ei kumpaakaan.

Jos latauspyyntö hyväksyttiin, käynnistää käyttäjän puolen koodi latauksen pilkkomalla lähetettävän videon 5 megatavun palasiin ja lähettämällä nämä yksi kerrallaan taustajärjestelmälle. Mikäli tiedosto löytyi jo palvelimelta, ei sitä ladata uudelleen ja lataus täten hylätään, mutta käyttäjälle kuitenkin ilmoitetaan, että prosessi on hyväksytty ja se siirretään jonoon odottamaan, että jokin Worker ottaa sen työn alle.

Kun lataus on valmis, ilmoitetaan käyttäjälle siitä, ja hän näkee My Videos -näkyvästä missä kohtaa prosessia hänen tekemänsä pyyntö tällä hetkellä menee (Queued, Decoding, Encoding, Post-processing, Ready). Riippuen järjestelmän ruuhkasta, siirtyy käyttäjän tekemä pyyntö joko jonoon odottamaan vapautuvaa Worker-säiettä, tai se otetaan vapaan Workerin tapauksessa suoraan työn alle. Tällä hetkellä järjestelmä järjestää pyynnöt, First In, First Served –periaattella, eli kaikki pyynnöt ennen tämänhetkistä pyyntöä tullaan käsittelemään ensin.

### 4.5.2 Videon enkoodaus/transkoodaus

Videon enkoodaus-/transkoodaus alkaa, kun jokin Worker-säikeistä vapautuu ja käyttäjän pyyntö on seuraavana jonossa.

Prosessi alkaa siten, että Worker hakee pyyntöön liittyvät tiedot tietokannasta ja selvittää, että miten pyyntö pitää esikäsitellä. Samalla käyttäjälle lähetetään viesti, että videota on aloitettu dekodamaan. Mikäli sisään tullut tiedosto on 8-bittisessä YUV 4:2:0 -formaattissa ei Workerin tarvitse tehdä tiedostolle yhtään mitään. Mikäli tiedosto on raakakuvaa jossain muussa kuin edellä mainitussa formaatissa, muuntaa Worker sen siihen formaattiin. Mikäli tiedosto on jossain säiliömuodossa (esim. mp4), purkaa Worker alkuperäisen videotiedoston YUV 4:2:0 -formaattiin. Tässä yhteydessä myös videon ääniraita otetaan talteen, mikäli sellainen alkuperäisestä videosta löytyy. Tässä prosessissa voi mennä pitkäänkin riippuen sisään tulleen tiedoston koosta.

Nyt video on oikeassa raakakuvaformaattissa ja se voidaan antaa Kvazaarille pakattavaksi. Kvazaarille annetaan myös mahdolliset käyttäjän määrittämät lisäparametrit ja pakkausasetus. Tässä yhteydessä käyttäjälle ilmoitetaan, että videon enkoodaus on aloitettu. Tämä on toinen hyvin aikaa vievä prosessi, riippuen tietenkin tiedostokoosta ja pakkausasetuksesta.

### 4.5.3 Videon muxaus

Kun enkoodaus on valmis, tarkistaa Worker minkäläistä ulostuloa käyttäjä halusi ja lähettää se käyttäjälle viestin, jossa kerrotaan, että ollaan siirretty *Post-processing*-vaiheeseen. Mikäli käyttäjä määrittä HEVC-ulostulon, siirtää Worker tiedoston *upload*-kansioon ja ilmoittaa käyttäjälle, että tiedosto on ladattavissa. Muussa tapauksessa se lisää HEVC-videon pyydettyyn säiliöformaattiin ja lisää videon alkuperäisen ääniraidan mukaan, mikäli sellainen alkuperäisessä videossa oli mukana. Tätä prosessia kutsutaan muxaukseksi (engl. muxing). Kun muxaus on valmis, siirretään valmis video *upload*-kansioon ja ilmoitetaan käyttäjälle, että prosessi on valmis.

#### 4.5.4 Videon tallennus

Kun video on valmis, siirretty oikeaan kansioon ja pyyntö poistettu pyyntöjonosta, ilmoittaa Worker käyttäjälle, että video on valmis ladattavaksi. Videonlatauksia on jokaista pyyntöä kohti kolme, jonka jälkeen video poistetaan taustajärjestelmän levyltä. Mikäli käyttäjä tarvitsee videota vielä kolmannenkin latauskerran jälkeen, joutuu hän luomaan uuden pyynnön.

Käyttäjä voi myös halutessaan poistaa pyynnön järjestelmästä. Tämä ei kuitenkaan poista alkuperäistä ladattua tiedostoa, vaan ainoastaan transkoodatun videon ja siihen liittyvän tiketin tiedokannasta.

## 5. TOTEUTETUN JÄRJESTELMÄN SUORITUSKYKY JA VERTAILU

Järjestelmän sisäinen suorituskyky, eli kuinka nopeasti järjestelmä kykenee transkoodaamaan videon, mitattiin pakkaamalla samaa videota vaihtelevilla kuormitusmäärillä. Tämän lisäksi vertailtiin vielä Kvazaar Cloud Encoderin ominaisuuksia suhteessa muihin pilvitranskooderiratkaisuihin.

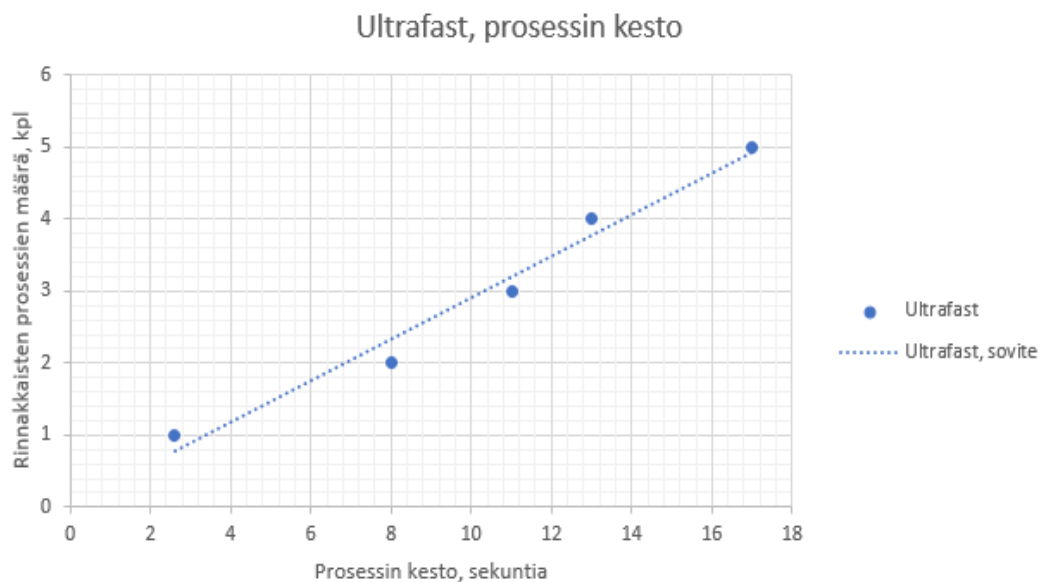
### 5.1 Suorituskyky

Järjestelmän arkkitehtuurin takia sen suorituskyky on suoraan verrannollinen siihen, kuinka paljon rinnakkaisia prosesseja järjestelmässä on. Järjestelmän suorituskyky mitattiin transkoodaamalla 1.6 MB H.264/AVC-videotiedosto.

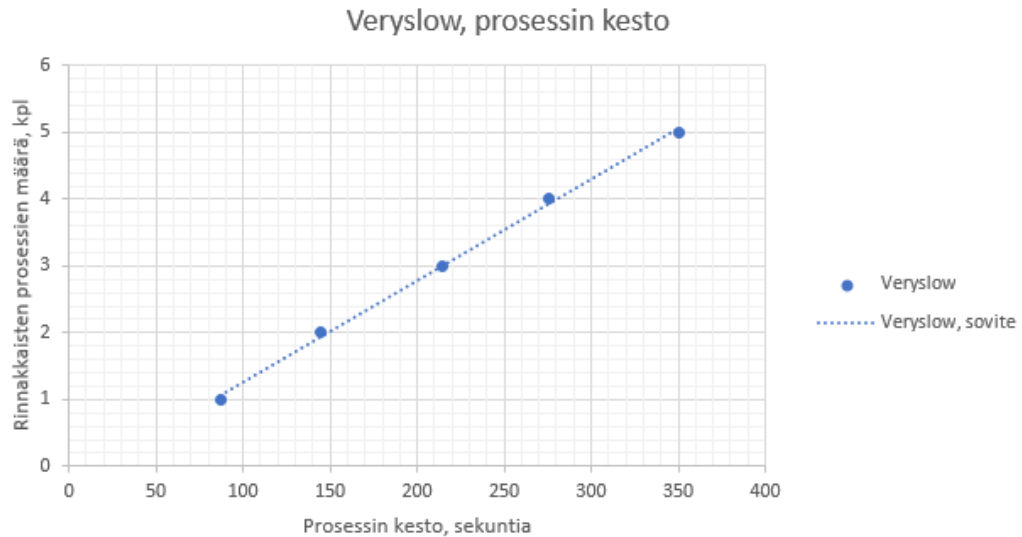
Nopeus mitattiin sekä ultrafast- että veryslow-preseteilla ja järjestelmää kuormitettiin lisäämällä rinnakkaisia taskeja. Testit suoritettiin tietokoneella, jossa oli 8-ytiminen, 3.40 GHz -kellotaajuudella pyörivä Intel i7-4770 –prosessori. Päämuistia oli 31 GB ja L1-välimuistin koko oli 8 kB. Taulukko 2 esittelee mittaustulokset. Kuvat 5 ja 6 esittävät mittaustulokset vielä graafisesti.

**Taulukko 2: Järjestelmän suorituskyky**

Prosessien määrä	1	2	3	4	5
ultrafast	2,6s	8s	11s	13s	17s
veryslow	87s	145s	214s	276s	350s



**Kuva 5: Järjestelmän suorituskyky, Ultrafast**



**Kuva 6: Järjestelmän suorituskyky, Veryslow**

Kuten kuvaajista voidaan huomata, prosessin kesto kasvaa lineaarisesti järjestelmän kuormituksen kasvaessa. Transkoodausprosessi on hyvin raskas ja testikoneen prosessorin kaikki ytimet pyörivät 100% kuormalla, kun prosesseja oli enemmän kuin kaksi. Yhden prosessin läpimenoaika pystyy pienentämään vähentämällä rinnakkaisten prosessien lukumäärää.

## 5.2 Vertailu aiempiin toteutuksiin

Tässä työssä esitelty Kvazaar Cloud Encoder -järjestelmä [39] sallii laajan skaalan eri sisään- ja ulostuloja ja pakkaa annetut videot HEVC-formaattiin. Toisin kuin luvussa 3.2 mainitut toteutukset, se on ilmainen käyttää, sen lähdekoodi on avointa ja vapaata sallien kenen tahansa muokata ja levittää koodia eteenpäin. Sen käyttöliittymä on helppokäyttöinen tarjoten kuitenkin mahdollisuuden kustomoida prosessia teknisesti edistyneemmälle käyttäjäkunnalle. Taulukko 3 esittelee oleellimmat ominaisuudet pilvitranskooderien välillä ja vertailee tunnetuimpia toteutuksia tässä työssä esiteltyyn järjestelmään.

**Taulukko 3: Pilvitranskooderien ominaisuuksien vertailu**

	Ilmainen käyttää	Avoin lähdekoodi	Raaka-sisääntulo	HEVC-ulostulo
Amazon	ei	ei	kyllä	kyllä
Coconut	ei	ei	ei	kyllä
Qencode	ei	ei	ei	kyllä
Zencoder	ei	ei	kyllä	kyllä
Cloud transcode	n/a	kyllä	ei	kyllä
Morph	n/a	kyllä	ei	ei
Snickers	n/a	kyllä	ei	kyllä
<b>Kvazaar Cloud Encoder</b>	kyllä	kyllä	kyllä	kyllä

Taulukossa 3 merkintä *n/a* tarkoittaa, että kriteeri ei päde kyseiseen pilvitranskooderiin. Esimerkiksi Morph ei ole missään vapaasti testattavissa, mutta sen koodin voi halutessaan ladata ja hostata järjestelmää itse.

## 6. YHTEENVETO

Videokuvan määrän räjähdysmäinen kasvu synnyttää tarpeen tehokkaalle pakkaukselle, mutta tällä hetkellä nämä työkalut eivät ole helppokäyttöisiä ja vaativat paljon pohjatietoa. Vaihtoehtoisesti monet pilvienkoodausjärjestelmät ovat maksumuurien takana tai niistä ei ole tehty mitään avoimia julkaisuja. Tässä kandidaatin työssä on esitelty käyttäjäystävällinen tapa pakata videokuvaa HEVC-formaattiin. Se pohjautuu FFmpeg-multimediatyökaluun sekä palkittuun Kvazaar-ohjelmistoon. Esitelty järjestelmän lähdekoodi on avointa ja sen lisenssi on hyvin permissiivinen mahdollistaen koodin muokkaamisen, jakamisen ja jopa myymisen. Järjestelmä on hyvin versatiili ja mahdollistaa peruskäyttäjille HEVC-pakkaamisen, mutta antaa myös kokeneemmille käyttäjille mahdollisuuden kustomoida pakkausprosessia erilaisilla parametreilla ja ulostuloilla.

Tulevaisuudessa Kvazaar Cloud Encoder -järjestelmän on tarkoitus tukea ennalta määriteltäviä pakkausasetuksia esimerkiksi 360-asteiselle videolle. Myös suoratoistetun videon transkoodausmahdollisuutta tullaan tutkimaan.

# LÄHTEET

- [1] High Efficiency Video Coding, document ITU-T Rec. H.265 and ISO/IEC 23008-2 (HEVC), ITU-T and ISO/IEC, Apr. 2013
- [2] Advanced Video Coding for Generic Audiovisual Services, document ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC), ITU-T and ISO/IEC, Mar. 2009
- [3] E. Dunic, M. Mustra, S. Grgic, G. Gvodzen, "Image Quality of 4:2:2 and 4:2:0 Chroma Subsampling Formats" in Proc. IEEE International Symposium ELMAR, Zadar, Croatia, Sept. 2009
- [4] H.264 is Magic. [Online]. Saatavissa: <https://sidbala.com/h-264-is-magic/> (viitattu 20.6.2019)
- [5] I. Bocharova "Compression for Multimedia". Cambridge, UK, University Press 2010 p. 2.
- [6] D.A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes", Proceedings of the I.R.E., September 1952, pp 1098-1102
- [7] Hirschberg, Daniel S., and S. Joseph Campanella. "Data Compression." *AccessScience*, McGraw-Hill Education, 2014.
- [8] Video compression and artifacts and MPEG noise reduction Saatavissa: <https://www.embedded.com/print/4013028> (viitattu 8.6.2019)
- [9] Supported Youtube Codecs [Online]. Saatavissa: <https://support.google.com/youtube/troubleshooter/2888402>
- [10] Netflix updates supported codecs [Online]. Saatavissa: <https://www.vdoCipher.com/blog/tech-update-netflix-updates-codecs-use-efficient-encoding/>
- [11] T. Wiegand, G. Sullivan, G. Bjontegaard, A. Luthra, "Overview of the H.264/AVC Video Coding Standard", *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 560-576, July 2003.
- [12] Kvazaar HEVC Encoder [Online] Available: <https://github.com/ultravideo/kvazaar>
- [13] M. Viitanen, A. Koivula, A. Lemmetti, A. Ylä-Outinen, J. Vanne, and T. D. Hämmäläinen, "Kvazaar: open-source HEVC/H.265 encoder," in Proc. ACM Int. Conf. Multimedia, Amsterdam, The Netherlands, Oct. 2016. DOI: <https://doi.org/10.1145/2964284.2973796>
- [14] Joint Collaborative Team on Video Coding Reference Software, HM [Online]. Saatavilla: <http://hevc.hhi.fraunhofer.de>
- [15] J. Xin, C. W. Lin, and M. T. Sun. Digital video transcoding. *IEEE*, 93(1):84-97, 2005.

- [16] I. Ahmad, X. Wei, Y. Sun, and Y.-Q. Zhang, "Video transcoding: an overview of various techniques and research issues," *IEEE Trans. on Multimedia*, vol. 7, no. 5, pp. 793-804, October 2005.
- [17] L. Wei, J. Cai, C. H. Foh, B. He, "QoS-aware resource allocation for video transcoding in clouds", *IEEE Trans. Circuits Syst. Video Technol.*, vol. 27, pp. 49-61, Jan. 2017.
- [18] Amazon Elastic Transcoder [Online]. Saatavissa: <https://aws.amazon.com/elastictranscoder> (viitattu 22.6.2016)
- [19] Coconut [Online]. Saatavissa: <https://coconut.co/> (viitattu 22.6.2019)
- [20] Qencode [Online]. Saatavissa: <https://cloud.gencode.com/> (viitattu 22.6.2019)
- [21] Brightcove Zencoder [Online] Saatavissa: <https://zencoder.com/en> (viitattu: 22.6.2019)
- [22] Z. H. Chang, B. F. Jong, W. J. Wong, and M. D. Wong, "Distributed video transcoding on a heterogeneous computing platform," in Proc. IEEE Asia Pacific Conf. Circuits Syst., Jeju, South Korea, Oct. 2016.
- [23] M. Chen, W. Chen, Z. Liu, and L. Cai, "Parallel video transcoding using Hadoop MapReduce," *Journal of Network Computing and Applications*, vol. 1, pp. 7-11, 2016.
- [24] Y. Dong, X. Zhang, Y. Zhao, and L. Song, "A containerized media cloud for video transcoding service," in Proc. IEEE Int. Conf. Consumer Electron., Las Vegas, NV, USA, Jan. 2018.
- [25] CloudTranscode [Online]. Saatavissa: <https://github.com/bfansports/CloudTranscode> (viitattu 8.6.2019)
- [26] G. Gao and Y. Wen. "Morph: a fast and scalable cloud transcoding system," in Proc. ACM Int. Conf. Multimedia, Amsterdam, The Netherlands, Oct. 2016. DOI: <https://doi.org/10.1145/2964284.2973792>
- [27] Snickers Video Encoder [Online]. Saatavissa: <https://github.com/snickers/snickers> (viitattu 8.6.2019)
- [28] jQuery [Online]. Saatavissa: <http://jquery.com/> (viitattu 22.6.2019)
- [29] Bootstrap [Online]. Saatavissa: <https://getbootstrap.com/> (viitattu 22.6.2019)
- [30] Node.js [Online]. Saatavissa: <https://nodejs.org/en/> (viitattu 22.6.2019)
- [31] FFmpeg [Online]. Saatavissa: <https://www.ffmpeg.org> (viitattu 22.6.2019)
- [32] Redis [Online]. Saatavissa: <https://github.com/antirez/redis> (viitattu 8.6.2019)
- [33] PostgreSQL [Online]. Saatavissa: <https://www.postgresql.org/> (viitattu 8.6.2019)
- [34] Docker [Online]. Saatavissa: <https://www.docker.com> (viitattu 8.6.2019)
- [35] WebSocket [Online]. Saatavissa: [https://developer.mozilla.org/en-US/docs/Web/API/WebSockets\\_API](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API) (viitattu 8.6.2019)

- [36] The WebSocket Protocol Saatavissa: <https://tools.ietf.org/html/rfc6455> (viitattu 8.6.2019)
- [37] Resumable.js [Online]. Saatavissa <http://www.resumablejs.com/> (viitattu 8.6.2019)
- [38] HTML5 [Online]. Saatavissa: <https://dev.w3.org/html5/html-author/> (viitattu 8.6.2019)
- [39] Kvazaar Cloud Encoder [Online]. Saatavissa: <https://github.com/ultra-video/cloud-encoder> (viitattu 8.6.2019)