Översikt av videoströmningsprotokoll med betoning på låg latens

Mathias Norräng 1800918
Kandidatavhandling i datateknik
Handledare: Sébastien Lafond
Fakulteten för naturvetenskap och teknik
Åbo Akademi
2021

Referat

Videoströmning blir allt vanligare idag, speciellt under Corona pandemin har videoströmning fått ett uppsving. Speciellt liveströmning blir allt populärare. Vid liveströmning kan det som tittare vara problematiskt om latensen mellan sändare och mottagare är för stor. De vanligaste strömningsprotokollen som används idag har väldigt hög latens på runt 20–30 sekunder. Vid liveevenemang kan det vara oacceptabelt högt. Det finns idag nya protokoll och tillägg kan användas för att sänka latensen mellan sändare och mottagare. I den här kandidatavhandlingen ges en översikt över dessa nya protokoll och tillägg.

Sökord: *Strömning, Låg latens, HLS, LL-HLS, MPEG-DASH, CMAF, WebRTC, SRT*

Innehållsförteckning

Introduktion 1

Allmänt om videoströmning 2

Strömning med adaptiv bittakt 3

Innehållsleveransnätverk 5

Dagens strömningsprotokoll 5

HLS 6

MPEG-DASH 9

Videoströmning i låg och ultralåg latens 12

Strömningsprotokoll som betonar låg latens 12

LL-HLS 13

CMAF 15

WebRTC 19

SRT 20

Diskussion 22

Referenser 23

Introduktion

Videoströmning är omåttligt populärt idag med streaming tjänster som till exempel YouTube och Netflix som visar beställvideo (Video on Demand) där kunden själv kan välja vad hen ser på och när. För att inte förlora sina tittare så flyttar även traditionell linjär television alltmer till webben för att kunna fortsätta nå ut till de som slutat se på tv [1].

Förutom strömning av förinspelat innehåll, blir även strömning av liveinnehåll allt vanligare när tittare ser på linjär tv via internet på sin dator eller mobil. När man tittar på linjär tv används oftast även en andraskärm (Second Screen). Där man tittar på till exempel sociala medier där andra tittare kanske kommenterar innehåll och resultat i realtid. Vid till exempel sportsändningar vill få veta resultatet innan de upplevt det själv.

Det finns idag även tjänster som specifikt specialiserar sig på distribution av användares liveströmmar som YouTube, Twitch och Facebook Gaming. På dessa plattformar är ofta publikens deltagande en stor del av upplevelsen. Sändaren vill ofta kunna kommunicera med sina tittare i realtid utan extra fördröjning. År 2022 kommer 17% av all internettrafik vara från liveströmning [2].

Även videochatt och videokonferensapplikationer, vars användning har exploderat i coronavirusets spår, använder strömmad video som är helt beroende av låg fördröjning mellan parterna.

Beroende på innehållet kan latensen mellan sändare och mottagare vara kritisk. I de flesta fall kan man acceptera latens på så mycket som 30–45 sekunder, men ibland behövs det kortare latens, det beror helt enkelt på användarfallet. Man kan kategorisera latensen i tre kategorier: reducerad latens, låg latens och ytterst låg latens [3], [4]. I den här avhandlingen kommer jag att beskriva och jämföra de olika sätt och tekniker som kan användas för att uppnå låg latens i videoströmning.

Allmänt om videoströmning

Det finns idag flera olika format och protokoll som kan användas när man ska skicka en videoström över internet. Vid utveckling av strömningslösningar väljer man det som passar ens användningsområde och vilken hårdvara och mjukvara som är tillgänglig; även innehållets natur måste tas i beaktande och det är också skillnad på förinspelat innehåll och direktsänt innehåll. Olika protokoll och format är olika bra i olika situationer.

De flesta konsumenter tar del av material över internet, idag sänds till och med ofta tv över internet (IPTV). Video strömmas idag oftast över HTTP-protokollet, genom att strömma över HTTP behöver man inte bry sig om paketförlust eller datadistorsion eftersom TCP-protokollet löser det problemet av sig själv. Detta lägger dock till extra pålägg (overhead) och kan även skapa latens. Genom att använda HTTP så kan strömmen även lätt passera eventuella brandväggar. Det motsatta fallet är video över UDP-protokollet där man har mindre pålägg och latens, men signifikant paketförlust och även andra problem om nätverksförhållandena inte är perfekta [1]. Nätverksförhållandena är nästan alltid problematiska med nätverk med betingad tjänstekvalitet (best effort).

Nätverk med betingad tjänstekvalitet är även ett problem för HTTP strömning. Därför har man utvecklat så kallad Adaptiv Bittakt (Adaptive Bit Rate Streaming, ABR) där video- och ljudströmmens kvalitet (bittakt) och således strömmen storlek ändras beroende på nätverksförhållandena och hastigheterna. Ofta tas även spelarens (enhetens) prestanda i beaktande i algoritmen. Adaptiv bittakt fungerar genom att sändaren strömmar flera olika versioner av video- och ljudströmmen med olika kvalitet; spelaren använder en ABR-algoritm för att räkna ut och välja den kvalitet som passar bäst. Detta för att videon både ska starta snabbt och till mån av möjlighet inte ska behöva buffra [1], [5].

För innehållsleverantörer spelar förutom en god strömnings upplevelse även innehållsskydd (DRM) en signifikant roll och är även en faktor vid val av strömningsformat. Idag finns två ”tävlande” standarder för

innehållkryptering CBCS och CTR. Att det finns två tävlande standarder har skapat ett problem. Vissa klienter är kompatibla med den ena standarden men inte den andra. Detta skapar problem med duplikation av källströmmar när man vill stödja de flesta klienter. Idag görs arbete att försöka konsolidera och förenkla innehållskryptering genom CENC (Common Encryption) [6].

De populäraste protokollen som används idag är MPEG:s Dynamic Adaptive Streaming over HTTP (DASH) och Apples HTTP Live Streaming (HLS) [7]–[9]. De protokollen passar väl vid överföring av förinspelat innehåll och direktsänt innehåll där latensen inte spelar någon direkt signifikant roll. När man dock ska strömma direktsänt innehåll där latensen spelar en viktig roll används ofta WebRTC eller ett eget format baserat på WebRTC. WebRTC sänds över UDP och kan därför åstadkomma en latens på under en sekund. Idag finns även nya protokoll som LL-HLS och CMAF som även de kan användas för att sänka latensen till en ofta acceptabel nivå.

Strömning med adaptiv bittakt

Allt eftersom flera börjat strömma innehåll över internet har behovet av adaptivitet i strömmen blivit allt större. I dagens läge finns flera olika sorters nätverk, allt från stabila fiberanslutningar till allt annat än stabila mobila nätverk. Genom att adaptivt ändra bittakten kan man stabilt strömma video även över annars ostabila nätverk. Idéen är att videon aldrig ska behöva buffra.

Strömning med adaptiv bittakt (ABR) har blivit populärt i och med att man börjat strömma video över HTTP. Det första populära protokollet att använda ABR var Microsofts Smooth Streaming och idag använder de största protokollen, HLS och MPEG-DASH, adaptiv bittakt.

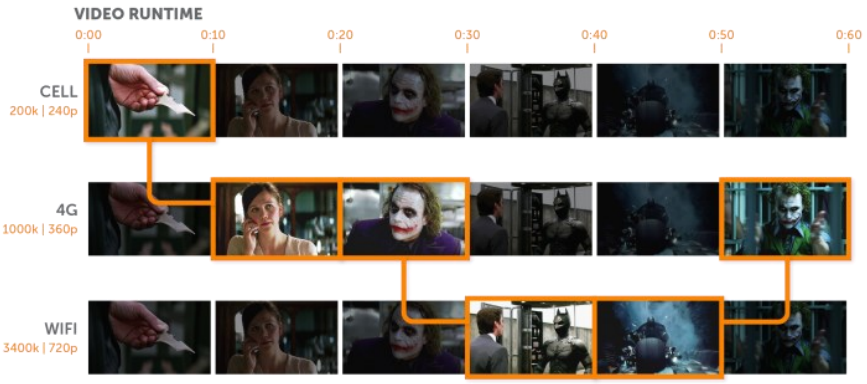
För att kunna distribuera en ström med adaptiv bittakt måste video- och ljudströmmen omkodas i olika bittakter, upplösningar och eventuellt även olika video- och ljudkodekar [10]. Omkodning är mycket resurskrävande och måste tas i beaktande speciellt vid liveströmning eftersom omkodningen måste ske i realtid för att strömmen ska fungera [11]. De olika

video- och ljudströmmarna måste sedan delas upp i mindre segment så att videospelaren kan byta mellan de olika kvalitéerna i farten [10].

Idag sker största delen av arbetet i spelaren. Det är den som väljer en passande resolution och bittakt efter en så kallad ABR-algoritm. Saker som ofta tas i beaktande när algoritmen beräknar passande kvalité är bland annat:

- Buffert storleken, algoritmen kontrollerar storleken av spelarens videobuffert.
- Nätverks hastighet och genomströmning, nästa segments kvalitet baseras på hur tidigare segment laddats ner.
- Skärmstorlek, om enheten har en låg skärmresolution behövs inte den högsta videoresolutionen.
- Spelarens bildhastighet och tappade bildrutor, om enheten inte klarar av att spela upp nuvarande kvalitet borde algoritmen välja en lägra kvalité.
- Antal tidigare kvalitets byten algoritmen gjort, om spelaren ofta behövt byta till en lägra algoritm skulle uppspelningsupplevelse troligtvis gynnas av en sämre kvalité men med färre byten.

Algoritmen använder dessa olika egenskaper för att adaptivt ändra kvalitet och skapa en smidig uppspelningsupplevelse. Figur 1 visar hur strömmen adaptivt kan ändra när nätverksförhållandena också gör det. Något som blivit allt viktigare när strömning blir alltmer populärt även utanför hemmet och via mobila nätverk.



Figur 1: Exempel på videouppspelning över ABR där kvaliteten ändrar beroende på nätverk

Källa: [9]

Innehållsleveransnätverk

Vid distribution av rika medier (Rich Media) är användningen av så kallade innehållsleveransnätverk (Content Delivery Network, CDN) nästan ett krav [1]. Speciellt då mängden klienter och även deras geografiska distribution ökar. Innehållsleveransnätverk används för att avlasta interna nätverk och serverapplikationer. Genom att använda ett innehållsleveransnätverk kan tjänstens bandbredds kostnader reduceras samtidigt som tillgängligheten ökar [12].

Innehållsleveransnätverk fungerar genom att ha en stor mängd servrar strategiskt spridda runt om hela världen [12], [13]. De är specifikt specialiserade på snabb dataöverföring och har ofta sina servrar direkt kopplade till internetknutpunkter (Internet Exchange Point, IXP) [12].

Ett innehållsleveransnätverk fungerar som cache och kan därför avlasta det interna nätverket. Detta gör att tjänsten klarar av att hantera mycket mera trafik och ger även systemet extra redundans och pålitlighet. Innehållsleveransnätverket används för att man ska kunna skala upp tjänsten [13].

Två kända innehållsleveransnätverk är Akamai och CloudFlare. De flesta molnleverantörer har även sina egna innehållsleveransnätverk.

Dagens strömningsprotokoll

Idag används främst två strömningsprotokoll HLS och MPEG-DASH, tidigare användes även Adobes RTMP och Microsofts Smooth Streaming men dessa har börjat fasas ut och används inte längre vid nya implementationer [8]. I de nuvarande protokollen har man i utformandet inte givit latensen så stor vikt utan mera fokuserat på uppspelningskvalitet och stabilitet.

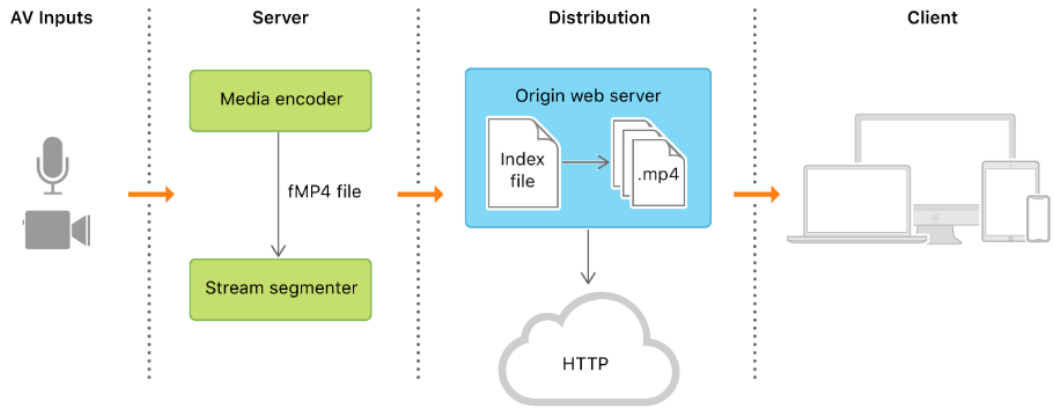
HLS

HTTP Live Streaming (HLS) är ett strömningsprotokoll skapat av Apple år 2009. När HLS skapades var Real-Time Messaging Protocol (RTMP) det populäraste strömningsprotokollet. RTMP är långt ett proprietärt och icke-öppet protokoll gjort av Macromedia och idag ägt av Adobe. RTMP kräver att man använder Flash Player. Då Apple valde att inte stödja Flash Player i sitt iOS ekosystem behövde de skapa ett nytt protokoll. HLS är idag det överlägset mest använda strömningsprotokollet [8] och kan användas i alla användarfall. Speciellt efter Flash Players nedgång har HLS blivit det föredragna protokollet för strömning över HTTP [8], [9].

HLS stödjer både liveströmning och beställvideo och kan användas på en bred mängd enheter [14]. Både iOS och Android har inbyggt stöd för HLS. Av skrivbordswebbläsarna är det idag endast Safari som har inbyggt stöd för HLS. I de andra webbläsarna kan ett JavaScript bibliotek som till exempel hls.js användas för att spela upp en HLS-ström [15].

Till skillnad från tidigare strömnings format stödjer HLS strömning av flera olika video- och ljudbittakter och upplösningar på samma gång. Detta tillåter ändring av bittakten som gensvar till ändringar i nätverksförhållandena genom adaptiv bittakt [9], [14]. HLS har även andra fördelar som stöd för undertexter i flera format (till exempel WebVTT och IMSC1), annonsinsättning, spolning framåt och bakåt, snabbspolning samt stöd för strömning av krypterad video [16], [17].

HLS är jämfört med tidigare strömningsformat väldigt simpelt och tekniskt sett inte ens en ström utan en serie progressiva nedladdningar. Genom progressiv nedladdning behöver en klient inte ladda ner hela videon före uppspelning kan börja [9], [18]. Videosegmenten strömmas från en helt vanlig webbserver. I och med detta kan man snabbt sprida videon över hela världen genom olika innehållsleveransnätverk. Detta gör att videoströmningslösningar med HLS enkelt och billigt kan skalas upp.



Figur 2: HLS strömnings scenario

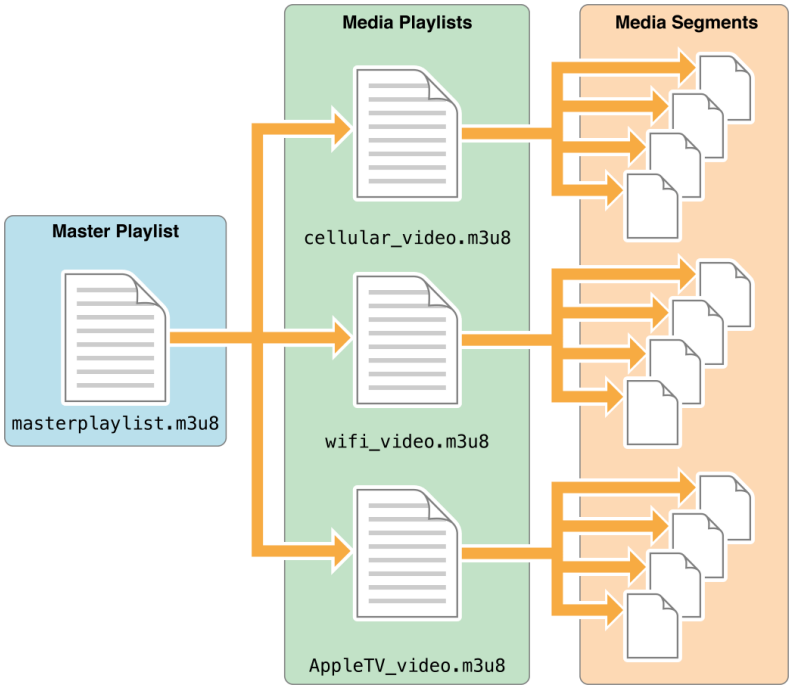
Källa: [19]

Ett HLS strömnings scenario kan se ut som i Figur 2. Det börjar med att en liveström eller en mediefil omkodas till flera olika så kallade *varianter* med olika upplösningar och bittakter. Upplösning och bittakt bör väljas så att en bred mängd enheter och nätverk stöds. Apple ger i HLS specifikationen rekommenderade värden på olika kvalitetsnivåer [16].

De olika *varianterna* segmenteras sedan till mindre mediesegment (media segment) som sparas tillsammans med en tillhörande mediespellista (media playlist). Från de olika mediespellistorna skapas sedan en huvudspellista (master playlist) som i Figur 3 [17]. De olika *varianterna* med tillhörande spellistor distribueras sedan genom en webbserver eller ett innehållsleveransnätverk [19]. Klienten laddar sedan ner huvudspellistan genom en URL till webbservern varvid klienten får tillgång till de olika segmenten och eventuella undertexter och dekrypteringsnycklar [9], [19]. Eftersom videon är uppdelad i segment kan klienten börja spela upp videon direkt när tillräckligt många segment laddats ner. Enligt HLS specifikationen krävs åtminstone tre segment innan uppspelning kan börja [19].

När video strömmas live genom HLS så har storleken på segmenten en större betydelse. Det eftersom uppspelning enligt specifikation inte kan börja före tre hela segment laddats ner. Segmentstorleken väljs vid kodningen och distributören kan själv välja segmentstorlek. Apples egna specifikationer rekommenderar en segmentstorlek på 6 sekunder [16]. Vid en segment storlek på 6 sekunder, skapas en latens mellan sändare och

mottagare på över 20 sekunder [9], [20]. Den här något höga latensen kan sänkas genom att använda kortare segment. Men det kan skapa instabilitet och idag kan i stället CMAF eller LL-HLS användas.



Figur 3: De olika kvalitetsnivåernas har egna spellistor.

Källa: [17]

HLS stöder endast videokodning i H.264 eller H.265. Video- och ljudströmmarna kan vara i en MPEG-TS eller fMP4 videobehållare [9], [16]. När HLS lanserades kunde endast MPEG-TS användas, men år 2016 började Apple även stödja fragmenterad MP4 (fMP4) [9] i och med CMAF samarbetet med Microsoft och MPEG. Idag är fMP4 det föredragna och rekommenderade videobehållarformatet [9], [21].

En HLS ström är alltså en serie ”.m3u8” spellistor i olika kvalitet. Spellistorna är textfiler som innehåller olika taggar. Figur 4 visar hur en HLS-huvudspellista med två kvalitetsnivåer kan se ut. Alla taggar som kan användas är specificerade i HLS-standarden [22].

Figur 4: En HLS huvudspellista för en beställvideoström med två kvalitetsnivåer.

Källa: [18]

MPEG-DASH

Dynamic Adaptive Streaming over HTTP eller MPEG-DASH är ett strömningsprotokoll skapat av MPEG (Motion Pictures Expert Group). MPEG-DASH har sin början i arbete gjord av 3GPP så tidigt som 2009. Idén bakom MPEG-DASH var att till skillnad från tidigare protokoll skapa en öppen och internationell standard för strömning över HTTP med adaptiv bittakt. Arbetet med MPEG-DASH var en kollaboration med flera företag och organisationer. Bland annat Microsoft, Netflix och Apple hjälpte till att utforma standarden. Arbetet med standarden (ISO/IEC 23009-1) färdigställdes år 2012 [23]. Idag är det DASH Industry Forum (DASH-IF) som har till uppgift att främja MPEG-DASH framtid och att utveckla formatet och dess användning [1].

MPEG-DASH har blivit det näst mest använda strömningsprotokollet efter HLS [8]. MPEG-DASH kan strömmas till de flesta enheter som stödjer HTML5 video och MSE (Media Source Extensions) [24]. Till skillnad från HLS så stöds inte MPEG-DASH på iOS och Apple TV [23]; detta är en klar motsättning för protokollet då kunder med iOS eller Apple TV inte stöds genom att endast använda MPEG-DASH.

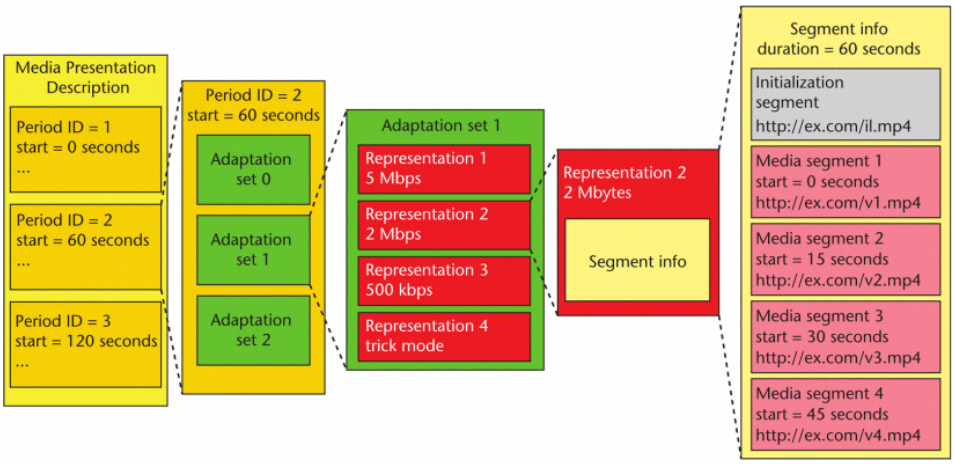
Likt HLS så tillåter MPEG-DASH strömning av olika upplösningar och video- och ljudbittakter med adaptiv bittakt. MPEG-DASH tillåter strömning av olika ljud-, video- och undertextspår som klienten direkt kan välja mellan. Det finns även funktionalitet för live annonsinsättning i strömmen.

MPEG-DASH stödjer även flertalet olika kopieringsskydds (DRM) plattformar, en MPEG-DASH ström kan till och med använda flera olika kopieringsskydds plattformar samtidigt, klienten laddar ner den video- och ljudström som stöds [25]. Flera stora plattformar som YouTube och Netflix använder MPEG-DASH för att strömma sitt innehåll [23].

En MPEG-DASH ström fungerar på ett liknande sätt som en HLS ström. Detta gäller då även arbetsflödet. För att skapa en MPEG-DASH ström omkodas en livestream eller mediefil till flera olika kvalitéer. Strömmen eller filerna segmenteras sedan och en Media Presentation Description (MPD) manifestfil skapas. MPD filen och tillhörande video-, ljud- och textfiler sparas sedan på en webbserver eller CDN. Klienten laddar ner MPD-filen varvid all information om strömmen fås. Klienten börjar ladda ner strömmen och uppspelning av videon börjar direkt när tillräckligt många segment laddats in.

I MPEG-DASH är segmentens storlek ofta runt 2–4 sekunder. Jämfört med HLS kan således en MPEG-DASH klient snabbare byta mellan kvalitéer i och med den korta segment storleken.

Kärnan i MPEG-DASH standarden är manifestfilen. Det är MPD-filen som definierar innehållet i en MPEG-DASH ström. MPD-filen är i XML format och innehåller alla olika kvalitéers ljud- och videospår och eventuella textspår och kopieringsskydd. Figur 5 visar hur en MPD är uppbyggd. MPD filen skapas automatisk av en segmenterare [25].



Figur 5: Strukturen av en MPD fil

Källa: [25]

MPEG-DASH är till sin design agnostisk då det gäller både video och ljud kodek. Till skillnad från HLS kan MPEG-DASH använda andra format utöver H.264 och H.265 [23], [26]. Populära alternativ är bland annat VP8 och VP9, två öppna format skapade av Google och som både YouTube och Netflix använder.

Strömning med MPEG-DASH använder främst fMP4 som videobehållare, enligt MPEG-DASH specifikation fungerar även MPEG-TS som videobehållare. Men de flesta spelare, inklusive referens spelaren dash.js, stödjer endast fMP4 [1], [27].

MPEG-DASH är till sin specifikation väldigt flexibelt, detta har blivit till både en för- och nackdel för protokollet. I och med den breda specifikationen så har det funnits problem med att skapa kodare och spelare som stödjer alla olika konfigurationer. Det har skadat interoperabiliteten i protokollet. DASH-IF har därför skapat en snävare rekommendation för att protokollet ska kunna fungera som förväntat på en bred mängd enheter och ekosystem [7]. ”DASH-AVC/264 Implementation Guidelines” som rekommendationen heter, specificerar bland annat att man i en MPEG-DASH ström måste använda videobehållaren fMP4, videokodeket H.264 och ljudkodeket HE-AAC v2 [28].

Videoströmning i låg och ultralåg latens

Det kan finnas många olika anledningar till att man vill ha låg latens. Oftast handlar det om att tittare vill kunna interagera med sändare eller att innehållet är tidskritiskt och latensen mellan sändare och mottagare måste vara lågt. Det kan bland annat handla om sport, nyheter, spel eller andra större evenemang som till exempel galor.

Videoprotokollet som används spelar i allra högsta grad en central roll. Det finns idag nya protokoll och tillägg till de gamla strömningsprotokollen (DASH och HLS). Tilläggen till de gamla strömningsprotokollen CMAF och LL-HLS kan användas för att sänka latensen mellan sändare och mottagare (end-to-end) till så lite som 1–3 sekunder över HTTP. Behöver man ännu lägre latens, under en sekund, så kan man använda WebRTC eller SRT beroende på användarfall.

Om ABR används måste man även noggrant välja ABR-algoritm. Orsaken till detta är att största delen av latensen oftast uppkommer främst i videobufferten i spelaren [5].

Det finns även problem som kan uppkomma när man strömmar med låg latens. Beroende på formatet kan distributionen av strömmen skalas dåligt, det är i alla fall fallet med WebRTC. DASH med CMAF och LL-HLS skalar dock mycket bättre eftersom de i grunden bara är vanliga MPEG-DASH eller HLS strömmar som lätt skalas med ett CDN.

Strömningsprotokoll som betonar låg latens

Länge var RTMP det enda protokollet för strömning med låg latens, idag finns flera olika alternativ.

Det finns ett behov av låglatenslösningar som billigt kan skalas upp. Under de senaste åren har stora förbättringar gjorts i de HTTP-baserade protokollen när det gäller latens. HLS har fått ett officiellt tillägg från Apple med LL-HLS som kan användas för att sänka den tidigare väldigt höga end-to-end latensen. Samarbetet mellan Microsoft, Apple och MPEG har även burit frukt med CMAF, ett nytt standardiserat format för segmenterad

medieleverans. Genom så kallad Chunked Transfer Encoding (CFE) kan CMAF användas tillsammans med till exempel MPEG-DASH för att sänka latensen [29].

LL-HLS

HLS har många fördelar som adaptiv bittakt, en god uppspelningskvalitet och möjligheten att använda kopieringsskydd. Men det finns en stor nackdel med HLS, den ytterst stora latensen. I och med HLS stora segment fås med en referenskonfiguration end-to-end latenser på runt 20–30 sekunder [9].

För att motverka problemet med höga latenser skapade Apple år 2019 LL-HLS. Det är inte ett nytt format utan snarare ett ”tillägg” till HLS och idag en del av HLS specifikationen. Detta gör att LL-HLS har alla de fördelar som HLS har, plus en kraftigt reducerad latens. LL-HLS använder vanliga HLS spellistor med några nya taggar. Genom LL-HLS kan man uppnå en latens på under 2 sekunder [30].

Det finns flera funktioner som LL-HLS använder för att sänka latensen. Huvudsakligen sänks latensen genom användning av mindre så kallade delsegment. Delsegmenten kan till exempel vara kodade i det nya CMAF formatet [31]. Delsegmenten skapas vid sidan om ett vanligt segment och läggs först till i spellistan. De är väldigt korta, ofta endast runt 200 millisekunder i längd. Genom att delsegmenten är så korta kan uppspelningen börja mycket snabbare än tidigare. Då tiden från videkodning till ett färdigt uppspelbart segment är endast 200 millisekunder. För att kunna hålla spellistans storlek nere tas delsegment bort när de är äldre än tre hela segment [29], [31].

Vid en liveströmning kan storleken på spellistan ändå bli väldigt stor, speciellt om liveströmmen håller på flera timmar. För att klienten ska slippa kontinuerligt ladda ner den allt större spellistan kan LL-HLS använda så kallade deltaspellistor. När spelaren börjar spela liveströmmen laddas huvudspellistan och mediespellistor ner. Efter det behöver spelaren endast ladda ner ändringar till spellistan genom deltaspellistor. Det här sparar på

bandbredd, speciellt då flera tusen samtidigt ska tittar på strömmen [29], [31].

LL-HLS lägger också till så kallade leveransdirektiv. Det är frågeparametrar som läggs på GET förfrågningar [31]. Leveransdirektiv gör det möjligt att blockera omladdningsbegäran av spellistaförfrågningar. Genom leveransdirektiv kan spelaren begära en spellista med segment som inte ännu finns, servern väntar då med att svara på förfrågan tills en spellista med segmentet finns att tillhandagå spelaren [31]. Blockering av spellistaförfrågningar kan minska belastningen på innehållsnätverk då dubblettförfrågningar kan förenas [29]. Förutom blockering av spellistaförfrågningar kan servern även direkt blockera nedladdningar av mediasegment. Tillsammans med så kallade ”förladdnings antydningar” (Preload Hints) i spellistan, kan spelaren fråga efter kommande delsegment. Servern väntar med att skicka delsegmenten tills de är tillgängliga [29].

Genom leveransdirektiven fås även förbättrad CDN-cachning när en ny spellista förfrågas. Leveransdirektiven är ”cache busting” och gör att innehållsleveransnätverket serverar en så ny version av spellistan som möjligt och inte en äldre cachad version [31]. Det här är viktigt när en ny klient ansluter och vill ha så låg latens som möjligt.

För att latensen ska kunna hållas låg, måste ändringar i nätverksförhållanden tas i beaktande så snabbt som möjligt. För att spelaren ska kunna övergå från en bittakt till en annan så smidigt som möjligt utan extra rundturer har varje kvalitets mediespellista även information om segment i de andra kvalitéerna [29], [31].

LL-HLS är i sin design fullt bakåtkompatibelt med HLS. Om en klient inte stödjer LL-HLS så spelas helt enkelt en vanlig HLS ström upp. Den låga latensen förloras men strömmen går ändå att titta på utan extra arbete.

När LL-HLS först presenterades, använde LL-HLS även HTTP/2 PUSH funktionalitet. Genom att använda HTTP/2 PUSH kunde protokollet slippa tur-och-returtiden vid hämtning av nya media segment [30]. Det här kravet togs bort av Apple för att protokollet skulle vara mera kompatibelt med nuvarande infrastruktur och innehållsleveransnätverk [29]. Genom att ta

bort kravet på HTTP/2 PUSH så har även kompatibiliteten med MPEG-DASH ökat.

Det bör även noteras att LL-HLS inte är samma sak som LHLS. LHLS är en tidigare låglatens version av HLS skapad av Periscope [9]. LHLS använder Chunked Transfer Encoding på samma sätt som CMAF för att uppnå låg latens. Apple valde att inte gå den vägen när de istället standardiserade LL-HLS [29].

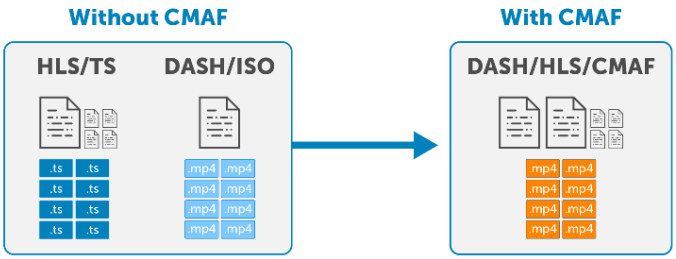
En stor fördel med LL-HLS är således att man kan uppnå latenser under 2 sekunder men fortfarande använda HLS. Att protokollet är bakåt-kompatibelt är även en bonus. Formatet är dock väldigt nytt och kompatibiliteten är inte ännu helt fullständig, både med tanke på innehållsnätverk och spelare. Även om LL-HLS kan skalas upp med hjälp av innehållsleveransnätverk sker mera förfrågningar. Det blir därför dyrare och mera påfrestande för både innehållsleveransnätverken och internetleverantörerna [29]. Eftersom formatet också är så nytt, så är det även okänt hur bra protokollet faktiskt presterar. Eftersom specifikationen är så ny så kan den även ändra ännu, speciellt med tanke på att Apple redan gjort ändring i den med till exempel borttagningen av HTTP/2 PUSH kravet.

CMAF

CMAF är inte i sig ett nytt strömningsprotokoll, utan en ny sorts behållare för video, ljud och undertexter [7], [32]. Vid CMAF:s utförande var målet att göra en utökbar kodningsstandard som kan användas oberoende av strömningsprotokoll [7]. Idag har både HLS och MPEG-DASH stöd för strömning av CMAF-formaterat innehåll [7], [33]. HLS fick stöd för CMAF i version 6 [34].

Idéen med CMAF skapades år 2016 av Microsoft och Apple tillsammans med MPEG och ett flertal stora företag i branschen [7]. CMAF är likt MPEG-DASH en öppen standard och internationell standard. Standarden (ISO/IEC 23000–19) blev klar och publicerades i januari 2018 [7].

Syftet med CMAF är att förena HLS och MPEG-DASH så att det inte behövs två eller flera kopior av samma video för olika protokoll. Genom att använda CMAF så sparas både tid och pengar vid distribution. Det här sker genom att endast koda video och ljud i ett format som kan användas av både HLS och MPEG-DASH, se Figur 6. Det hjälper signifikant med innehållsleveransnätverkens effektivitet och cachning [7]. Speciellt stora besparingar fås vid liveströmning. Vidare har CMAF dessutom stöd för både CENC- och CBCS-kryptering och genom att använda *Common Encryption* kan flera DRM-system användas samtidigt i samma fil [7].



Figur 6: CMAF gör det möjligt för HLS och MPEG-DASH att använda samma videofil.

Källa: [35]

CMAF skapar en bas för vidare samarbete oberoende manifest i framtiden. Speciellt då CMAF hjälper till med samarbete över hela strömningsbranschen. Det här märks redan med samarbetet mellan Apple och Microsoft.

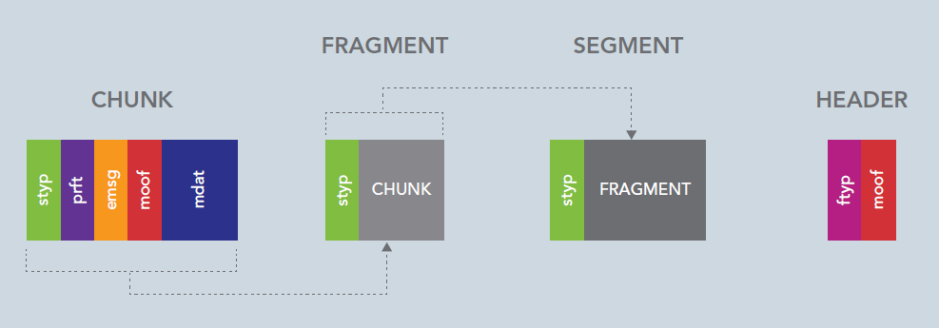
CMAF-formatet definierar medieprofiler och presentationsprofiler. Medieprofilerna används för att identifiera video, ljud och undertexter. De hjälper till med interoperabiliteten då de lägger krav på kodning och avkodning. En presentationsprofil skapas sedan utav de valda medieprofilerna. Det är medie- och presentationsprofilerna som gör CMAF utökninsbart då nya kodekar uppkommer, kan nya medieprofiler standardiseras [7].

CMAF använder ISO-BMFF (ISO Base Media File Format) formatet fMP4 som grund [2], [7], [33]. En CMAF ström delas upp i segment, i CMAF är segmenten oftast runt 6–10 sekunder. Segmenten kan i sin tur vidare delas upp i en eller flera mindre fragment [7], [32]. Ett fragment skall tillsammans

med en ”CMAF Header” vara en uppspelbar helhet. För att hålla videokodningen så effektiv som möjligt bör fragmenten vara minst några sekunder långa [7].

Varje fragment kan sedan ännu delas upp i mindre så kallade bitar (Chunk), bitarna innehåller själva mediasamplen [7], [32]. Bitarna behöver till skillnad från fragmenten inte vara individuellt uppspelbara, man kan i extremfall till och med ha en bildruta per bit [7]. Uppdelningen av en CMAF-ström syns tydligt i Figur 7.

En CMAF-ström kan innehålla flera olika spår (Track). Spåren kan användas för att strömma olika innehålls varianter, till exempel alternativa kvalitéer, kodekar eller språk. Byte av spår kan ske först när ett nytt fragment laddas in. I CMAF så måste varje spår vara en skild fil. CMAF tillåter alltså inte multiplex-fMP4, där till exempel video- och ljudspår finns i samma fil [7].



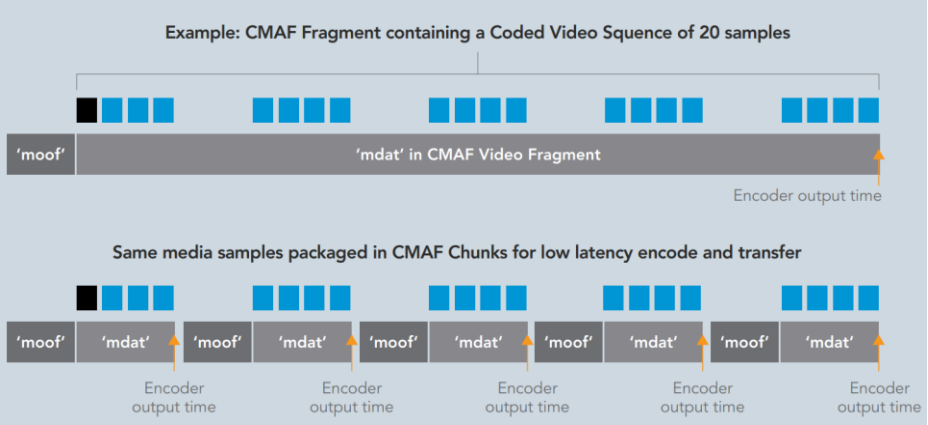
Figur 7: CMAF:s uppdelning i mindre delar.

Källa: [32]

CMAF har även speciella fördelar vid liveströmning. HTTP-baserad videoströmning har som noterat ett problem med hög latens. Genom att använda CMAF tillsammans med så kallad *Chunked Encoding* kan end-to-end latensen märkbart sänkas.

Chunked Encoding fungerar genom att bryta upp segmenten i flera mindre bitar (Chunks) som mycket snabbare kan propagera upp från kodare och segmenterare till innehållsnätverk, se Figur 8. Genom CMAF och *Chunked Encoding* kan spelaren snabbare börja spela upp strömmen vilket leder till lägre latens [2]. Idag är bitarna som används vid *Chunked Encoding* oftast 1–15 bildrutor långa. Genom att även använda HTTP/1.1 Chunked Transfer

behöver spelaren endast fråga efter första biten i ett segment, servern skickar sedan automatiskt resten till spelaren utan att den behöver göra extra tidsödslande förfrågningar [32]. I CMAF likt LL-HLS så skickas bitar endast för det nuvarande ofärdiga segmentet. När ett segment är klart så kasseras de enskilda bitarna [7]. Genom att använda *Chunked Encoding* så spelar inte segmentstorleken någon större roll [32]. Det går med CMAF och *Chunked Encoding* att sänka latensen till runt 2–6 sekunder. En latens som i de flesta fall är fullständigt acceptabel.



Figur 8: Jämförelse mellan en ström med och utan Chunked Encoding

Källa: [32]

CMAF har således två stora fördelar. Att det förenar strömningsprotokoll och -format. Redan idag kan samma ström användas för strömning med både HLS och DASH, något som dramatiskt minskar pålägg. Och att det via *Chunked Encoding* och *Chunked Transfer* dramatiskt går att sänka end-to-end latensen. Genom CMAF kan video samtidigt strömmas med låg latens och hög kvalitet som 4K-video med hög bildfrekvens [36]. CMAF:s standardisering och utökbarhet är även positivt, till exempel stöd för nya förbättrade kodekar när de kommer ut. Det gör att CMAF kanske kan bli det ”sista formatet” vid liveströmning.

WebRTC

WebRTC är ett fritt och öppet ramverk för realtidskommunikation mellan webbläsare över internet. WebRTC fungerar helt genom webbläsaren utan att behöva använda till exempel Flash [37]. WebRTC är utvecklat under IETF (Internet Engineering Task Force) och W3C (World Wide Web Consortium) [38], [39]. Själva protokollet definieras i ett antal IETF-standarder medan webbläsarnas programmeringsgränssnitt standardiseras av W3C [39]. Den officiella standarden blev färdig i januari 2021 [40], [41].

Hela grundidén med WebRTC var realtids röst- och ljudkommunikation mellan webbläsare i form av videochat- och konferensapplikationer. Men WebRTC är väldigt flexibelt och kan även användas för liveströmning av en video- och ljudström. Protokollet är välutformat för strömning till mindre grupper [37] med strömning från en till en (one-to-one) eller en till få (one-to-few). Till skillnad från de tidigare protokollen är WebRTC en bitström som huvudsakligen strömmas över UDP (även om TCP också är möjligt). WebRTC är genom sin design låglatent och latenser under 500 millisekunder är nåbara [37].

Det finns dock problem som gör att WebRTC inte alltid är en perfekt eller praktisk lösning. Det skalas bland annat ganska dåligt och blir därför snabbt dyrt. Det finns speciella innehållsleveransnätverk för WebRTC med de är få och dyra [37], [42]. Det finns även problem med inkonsekvent implementering i olika webbläsare och olika webbläsare stöder olika videokodekar. WebRTC kräver även användningen av en specialiserad medieserver eller signalserver för att fungera och kunna slippa förbi eventuella brandväggar [37].

Med WebRTC går det egentligen inte heller att strömma med adaptivbittakt och en WebRTC ström kan inte heller uppnå samma ”broadcast” kvalitet som de andra protokollen [42]. Strömmen kan lida i kvalitet och kan även vara något ostabil vid överföring över UDP. Vid strömning till en stor mängd personer så passa till exempel LL-HLS mycket bättre.

SRT

Secure Reliable Transport (SRT) är ett relativt nytt strömningsprotokoll. Likt WebRTC så är SRT ett UDP-baserat protokoll. Det är skapat av Haivision och det var först proprietärt; men för att öka antagandet av protokollet så valde Haivision att öppna upp SRT och det är idag öppen källkod [43], [44].

Idéen med SRT är att det ska används ”den första milen”. SRT används inte för att skicka video till konsumenter utan snarare från inspelningsplats till medieserver. SRT kan användas för att ersätta äldre protokoll som RTSP och Adobes RTMP; två protokoll som till viss del har börjat visa sin ålder [45].

SRT har flera stora fördelar jämfört med till exempel RTMP. SRT är ”innehålls” agnostiskt. Det stöder videoströmning, men kan även användas för helt annan dataströmning [43]. I och med det så stöds alla videoformat inklusive H.265, ett format som RTMP inte stödjer. SRT stödjer även överföring av flera olika medieströmmar samtidigt, över samma anslutning, även om strömmarna är av helt olika karaktär [43].

Det klarar även av dåliga nätverksförhållanden exceptionellt bra. Strömningsprotokollet gör det möjligt att återsända ”försvunna” paket och använda felkorrigering för att fixa fel i strömmen. Det här är speciellt viktigt vid strömning över nätverk med betingad tjänstekvalitet [43], [45]. Figur 9 visar hur samma ström kan se ut när den sänds över UDP och SRT.

Figur 9: Samma videoström över UDP och SRT vid en paketförlust på 2%

Källa: [43]

SRT är utformat utan att kräva någon central server, men så att det ändå enkelt slipper igenom eventuella brandväggar. När en STR ström startar så sker en handskakning. SRT har tre lägen, en server kan vara i uppringar- (Caller), lyssnar- (Listener) eller i rendezvousläge. Rendezvousläget kan användas för att lättare slippa igenom brandväggar. Vid strömning med SRT kan strömmen eventuellt krypteras med AES 128-, 192- eller 256-bitars kryptering [43], [45]. Protokollet är även mera effektivt än RTMP. Med SRT går det att få ut mera av samma nätverksbandbredd jämfört med RTMP. Även de latenser som kan åstadkommas är lägre med SRT. Latenser under 500 ms. är uppnåbara [45], [46].

Diskussion

Videoströmning är idag omåttligt populärt, speciellt i dessa Corona tider. Vi ser idag på mera rörligt innehåll än någonsin förut, både strömmat och förinspelat. Och allt det sker över internet; idag sänds till och med tv ofta över internet. När man ser på strömmad video används ofta någon sorts andraskärm. Då vill man inte få upplevelsen förstörd genom spoilers via till exempel sociala medier. För att minska den risken borde latensen mellan sändare och mottagare vara så låg som möjligt och det är något dagens format och protokoll inte hjälper med. Idag finns som tur nya protokoll vars mål specifikt är att sänka latensen.

Jag ser särskilt positivt på CMAF. Genom att byta till CMAF formatet kan man med hjälp av Chunked Encoding uppnå latenser på under tre sekunder. Lyko m.fl. har även visat att man bara genom att byta till CMAF kunde få en reduktion av om-buffringar på 43–71% [5]. CMAF har även andra användningsområdena utanför låglatensströmning. Det att man med CMAF kan använda samma videoformat för både DASH och HLS är även positiv då man inte måste ha ett skilt protokoll för iOS. Om man vill ha låg latens med ett öppet protokoll så passar definitivt CMAF väl. Med två år bakom sig och flera kommersiella- och öppen-källkodsimplementationer så är CMAF definitivt ett passande val. LL-HLS har också visat sig vara användbart i kampen mot latens. Med under två sekunders end-to-end latens är LL-HLS även ett bra val även om LL-HLS är nyare och kanske inte ännu helt moget.

Sen finns det även andra protokoll som WebRTC med en latens på under en sekund. Man bör dock tänka på om man faktiskt behöver en så låg latens. I de flesta fall räcker LL-HLS eller LL-DASH (MPEG-DASH med CMAF) bra till. Man kan även använda så kallade hybrid lösningar, där en del har tillgång till en ultra-låglatent ström och andra som inte behöver det kan se på en vanlig ström.

Jag har i den här avhandlingen förhoppningsvis kunnat ge en något klar översikt över de strömningsprotokoll som används idag, både nya och gamla. Eftersom valet av strömningsprotokoll har så stor inverkar på latensen är det viktigt att välja ett format med rätt förutsättningar.

Referenser

[1] B. Bing, *Next-Generation Video Coding and Streaming*. Hoboken: John Wiley & Sons, Incorporated, 2015.

[2] R. Viola, A. Gabilondo, A. Martin, J. F. Mogollón, M. Zorrilla, och J. Montalbán, ”QoE-based enhancements of Chunked CMAF over low latency video streams”, i *2019 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, juni 2019, s. 1–6, doi: 10.1109/BMSB47279.2019.8971894.

[3] ”Video Latency in Live Streaming - Amazon Web Services”, *Amazon Web Services, Inc.* https://aws.amazon.com/media/tech/video-latency-in-live-streaming/ (åtkomstdatum mar. 21, 2021).

[4] ”What Is Low Latency and Who Needs It?”, *Wowza Media Systems*, juli 09, 2020. https://www.wowza.com/blog/what-is-low-latency-and-who-needs-it (åtkomstdatum mar. 21, 2021).

[5] T. Lyko, M. Broadbent, N. Race, M. Nilsson, P. Farrow, och S. Appleby, ”Evaluation of CMAF in live streaming scenarios”, i *Proceedings of the 30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, Istanbul Turkey, juni 2020, s. 21–26, doi: 10.1145/3386290.3396932.

[6] ”What is CMAF and the implication for HLS”, *CDN.net*, maj 21, 2019. https://cdn.net/what-is-cmaf-and-the-implication-for-hls/ (åtkomstdatum mar. 27, 2021).

[7] A. C. Begen och Y. Syed, ”Are The Streamingformat Wars Over?”, i *2018 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, juli 2018, s. 1–4, doi: 10.1109/ICMEW.2018.8551563.

[8] ”Bitmovin Video Developer Report 2019”. Bitmovin, 2019, Åtkomstdatum: mar. 21, 2021. [Online]. Tillgänglig vid: https://go.bitmovin.com/video-developer-report-2019.

[9] ”What Is HLS Protocol (HTTP Live Streaming)”, *Wowza Media Systems*, maj 07, 2020. https://www.wowza.com/blog/hls-streaming-protocol (åtkomstdatum mar. 21, 2021).

[10] ”Adaptive Bitrate Streaming: How It Works and Why It Matters”, *Wowza Media Systems*, feb. 09, 2021. https://www.wowza.com/blog/adaptive-bitrate-streaming (åtkomstdatum mar. 27, 2021).

[11] ”What Is Transcoding and Why Is It Critical for Streaming?”, *Wowza Media Systems*, dec. 31, 2019. https://www.wowza.com/blog/what-is-transcoding-and-why-its-critical-for-streaming (åtkomstdatum mar. 27, 2021).

[12] ”What is a CDN? | How do CDNs work?”, *Cloudflare*. https://www.cloudflare.com/learning/cdn/what-is-a-cdn/ (åtkomstdatum mar. 31, 2021).

[13] ”What Is a CDN and Why Is It Critical to Live Streaming?”, *Wowza Media Systems*, jan. 21, 2020. https://www.wowza.com/blog/cdn-live-streaming (åtkomstdatum mar. 23, 2021).

[14] ”HTTP Live Streaming”. https://developer.apple.com/documentation/http_live_streaming (åtkomstdatum mar. 21, 2021).
[15] *video-dev/hls.js*. video-dev, 2021.
[16] ”HLS Authoring Specification for Apple Devices”. https://developer.apple.com/documentation/http_live_streaming/hls_authoring_specification_for_apple_devices (åtkomstdatum mar. 21, 2021).
[17] ”About HTTP Live Streaming”. https://developer.apple.com/library/archive/referencelibrary/GettingStarted/AboutHTTPLiveStreaming/about/about.html (åtkomstdatum mar. 21, 2021).
[18] A. Begen, T. Akgul, och M. Baugher, ”Watching Video over the Web: Part 1: Streaming Protocols”, *IEEE Internet Comput.*, vol. 15, nr 2, s. 54–63, mar. 2011, doi: 10.1109/MIC.2010.155.
[19] ”Understanding the HTTP Live Streaming Architecture | Apple Developer Documentation”. https://developer.apple.com/documentation/http_live_streaming/understanding_the_http_live_streaming_architecture (åtkomstdatum mar. 21, 2021).
[20] ”Example Playlists for HTTP Live Streaming”. https://developer.apple.com/documentation/http_live_streaming/example_playlists_for_http_live_streaming (åtkomstdatum mar. 22, 2021).
[21] ”Deploying a Basic HTTP Live Stream”. https://developer.apple.com/documentation/http_live_streaming/deploying_a_basic_http_live_stream (åtkomstdatum mar. 21, 2021).
[22] R. Pantos, ”HTTP Live Streaming 2nd Edition”. https://tools.ietf.org/html/draft-pantos-hls-rfc8216bis-08 (åtkomstdatum mar. 29, 2021).
[23] ”MPEG-DASH: Dynamic Adaptive Streaming Over HTTP Explained”, *Wowza Media Systems*, feb. 25, 2021. https://www.wowza.com/blog/mpeg-dash-dynamic-adaptive-streaming-over-http (åtkomstdatum mar. 23, 2021).
[24] ”Setting up adaptive streaming media sources - Developer guides”. https://developer.mozilla.org/en-US/docs/Web/Guide/Audio_and_video_delivery/Setting_up_adaptive_streaming_media_sources (åtkomstdatum mar. 23, 2021).
[25] I. Sodagar, ”The MPEG-DASH Standard for Multimedia Streaming Over the Internet”, *IEEE Multimed.*, vol. 18, nr 4, s. 62–67, apr. 2011, doi: 10.1109/MMUL.2011.71.
[26] ”What is MPEG-DASH?”, *Cloudflare*. https://www.cloudflare.com/learning/video/what-is-mpeg-dash/ (åtkomstdatum mar. 23, 2021).
[27] ”Mpeg-ts in Dash.js · Issue #483 · Dash-Industry-Forum/dash.js”, *GitHub*. https://github.com/Dash-Industry-Forum/dash.js/issues/483 (åtkomstdatum mar. 23, 2021).
[28] ”DASH Industry Forum | Catalyzing the adoption of MPEG-DASH”. https://dashif.org/about/ (åtkomstdatum mar. 24, 2021).
[29] K. Durak, M. N. Akcay, Y. K. Erinc, B. Pekel, och A. C. Begen, ”Evaluating the Performance of Apple’s Low-Latency HLS”, i *2020*

IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP), sep. 2020, s. 1–6, doi: 10.1109/MMSP48831.2020.9287117.
[30] ”Apple Low-Latency HLS: What It Is and How It Relates to CMAF | Wowza”, *Wowza Media Systems*, feb. 03, 2020. https://www.wowza.com/blog/apple-low-latency-hls (åtkomstdatum mar. 26, 2021).
[31] ”Enabling Low-Latency HLS”. https://developer.apple.com/documentation/http_live_streaming/enabling_low-latency_hls (åtkomstdatum mar. 26, 2021).
[32] W. Law, ”Ultra-Low-Latency Streaming Using Chunked-Encoded and Chunked-Transferred CMAF”. Akamai, okt. 2018, Åtkomstdatum: apr. 04, 2021. [Online]. Tillgänglig vid: https://www.akamai.com/es/es/multimedia/documents/white-paper/low-latency-streaming-cmaf-whitepaper.pdf.
[33] ”CMAF – What it is and why it may change your OTT future.” https://community.akamai.com/customers/s/article/CMAF-What-it-is-and-why-it-may-change-your-OTT-future?language=en_US (åtkomstdatum apr. 04, 2021).
[34] ”Video: What Are the Key Features of CMAF?”, *Streaming Media Magazine*, juni 07, 2019. https://www.streamingmedia.com/Articles/ReadArticle.aspx?ArticleID=132063&utm_source=related_articles&utm_medium=gutenberg&utm_campaign=editors_selection (åtkomstdatum apr. 03, 2021).
[35] ”About CMAF workflows”. https://www.wowza.com/docs/about-cmaf-workflows-in-wowza-streaming-engine (åtkomstdatum apr. 04, 2021).
[36] ”What Is CMAF (Common Media Application Format)”, *Wowza Media Systems*, jan. 23, 2019. https://www.wowza.com/blog/what-is-cmaf (åtkomstdatum apr. 04, 2021).
[37] ”Webinar: WebRTC Tutorial”, *Wowza Media Systems*, juli 31, 2020. https://www.wowza.com/blog/webinar-webrtc-tutorial (åtkomstdatum apr. 07, 2021).
[38] F. Rhinow, P. P. Veloso, C. Puyelo, S. Barrett, och E. O. Nuallain, ”P2P live video streaming in WebRTC”, i *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, jan. 2014, s. 1–6, doi: 10.1109/WCCAIS.2014.6916588.
[39] Shuai Zhao, Zhu Li, och D. Medhi, ”Low delay MPEG DASH streaming over the WebRTC data channel”, i *2016 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, juli 2016, s. 1–6, doi: 10.1109/ICMEW.2016.7574765.
[40] ”Web Real-Time Communications (WebRTC) transforms the communications landscape; becomes a World Wide Web Consortium (W3C) Recommendation and multiple Internet Engineering Task Force (IETF) standards”. https://www.w3.org/2021/01/pressrelease-webrtc-rec.html.en (åtkomstdatum apr. 07, 2021).
[41] ”WebRTC 1.0: Real-Time Communication Between Browsers”. https://www.w3.org/TR/webrtc/ (åtkomstdatum apr. 07, 2021).
[42] Wowza Media Systems, *Webinar: Low-Latency Delivery — LL-HLS vs. WebRTC*. 2020.

[43] ”SRT Open Source Streaming for Low Latency Video”, *Haivision*. https://www.haivision.com/resources/white-paper/srt-open-source/ (åtkomstdatum apr. 06, 2021).
[44] ”SRT - FAQ”, *SRT Alliance*. https://www.srtalliance.org/srt-faq/ (åtkomstdatum apr. 06, 2021).
[45] R. Viola *m.fl.*, ”Adaptive Rate Control for Live streaming using SRT protocol”, i *2020 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, okt. 2020, s. 1–6, doi: 10.1109/BMSB49480.2020.9379708.
[46] ”RTMP vs. SRT: Comparing Latency and Maximum Bandwidth”, *Haivision*. https://www.haivision.com/resources/white-paper/srt-versus-rtmp/ (åtkomstdatum apr. 06, 2021).