UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

"STUDIO DELLE TECNICHE PIÙ RECENTI DI LIVE STREAMING"

Relatore: Prof. Marco Cagnazzo

Laureando: Carlo Bottaro

ANNO ACCADEMICO 2023 – 2024

Data di laurea 15/07/2024

Sommario

Negli ultimi anni, il live streaming è diventato fondamentale per la comunicazione digitale, utilizzato in vari ambiti come l'intrattenimento, l'istruzione, lo sport e il lavoro. Questa tesi si propone di esplorare e analizzare le tecniche più recenti nel campo del live streaming, concentrandosi sulle tecnologie emergenti e sulle strategie di trasmissione in tempo reale. L'obiettivo principale di questo lavoro è fornire una panoramica chiara e dettagliata del significato di streaming e live streaming, della loro storia e delle tecnologie che li hanno resi la grande fonte di intrattenimento e opportunità lavorative che sono nel mondo moderno

Indice

Elenco delle figure

Lista di abbreviazioni

ABR *Adaptive Bitrate Streaming.*
AMF *Action Message Format.*
API *Application Programming Interface.*

CDN *Content Delivery Network.*
CTE *Chunked Transfer Encoding.*

DASH *Dynamic Adaptive Streaming over HTTP.*
DCT *Discrete Cosine Transform.*
DLR *Deep Reinforced Learning.*
DNN *Deep Neural Network.*

EC *Edge Computing.*

FFN *Feed Forward Network.*

G2G *Glass to Glass.*

HLS *HTTP Live Streaming.*
HTTP *Hypertext Transfer Protocol.*

ICE *Interactive Connectivity Establishment.*
IETF *Internet Engineering Task Force.*
IF *Influence Factor.*
IoT *Internet of Things.*
ISP *Internet Service Provider.*
ITU *International Telecommunication Union.*

LL-DASH *Low-Latency DASH.*
LL-HLS *Low-Latency HTTP Live Streaming.*
LS *Live Streaming.*

MEC *Multi-access Edge Computing.*
MOS *Mean Opinion Score.*
MPD *Multimedia Presentation Description.*
MPEG *Moving Picture Experts Group.*

MSE *Mean Square Error.*

NAT *Network Address Translation.*

PoP *Point of Presence.*

PSNR *Peak Signal-to-Noise Ratio.*

QoE *Quality of Experience.*

QoS *Quality of Service.*

QUIC *Quick UDP Internet Connections.*

RLHF *Reinforced Learning with Human Feedback.*

RTMP *Real-Time Messaging Protocol.*

RTP *Real-Time Transport Protocol.*

RTSP *Real-Time Streaming Protocol.*

RTT *Round Trip Time.*

SDP *Session Description Protocol.*

SRTP *Secure Real-Time Transport Protocol.*

SSIM *Structural Similarity Index Measure.*

STUN *Session Traversal Utilities for NAT.*

TCP *Transmission Control Protocol.*

TLS *Transport Layer Security.*

TURN *Traversal Using Relay around NAT.*

UDP *User Datagram Protocol.*

URI *Uniform Resource Identifier.*

W3C *World Wide Web Consortium.*

WebRTC *Web Real-Time Communication.*

Capitolo 1

Introduzione

Oggigiorno il *Live Streaming* (LS) video ha assunto un ruolo sempre più centrale, nel 2023 quasi il 20% del traffico internet mondiale è stato utilizzato per guardare LS[1], percentuale in aumento da anni, soprattutto negli anni successivi alla pandemia COVID-19. Con l'evoluzione delle tecnologie di rete e l'aumento della larghezza di banda disponibile, è diventato possibile trasmettere eventi in tempo reale a un pubblico globale, con una qualità video sempre maggiore. Tuttavia, la trasmissione in tempo reale pone una serie di sfide tecniche, tra cui la gestione della latenza, la *Quality of Service* (QoS) e la *Quality of Experience* (QoE). Questa tesi si propone di esplorare le tecniche più recenti e innovative nel campo del LS, con un focus finale sulle tecnologie emergenti e le strategie di trasmissione in tempo reale.

1.1 Contesto e Motivazione

Lo streaming è una tecnologia che permette la trasmissione di dati multimediali (audio e video) ,attraverso una rete cablata o senza fili, in modo che i contenuti possano essere riprodotti od utilizzati mentre il resto dei dati è ancora in fase di trasferimento, eliminando così la necessità di scaricare interamente il file prima della riproduzione. Lo streaming può essere in real-time o live. Si parla di real-time streaming se il processo di trasmissione dei dati avviene con un ritardo trascurabile, generalmente misurato in millisecondi consentendo l'utilizzo dei dati contemporaneamente alla trasmissione. Lo streaming in tempo reale viene utilizzato in casi in cui è richiesta una risposta immediata agli eventi, ad esempio nel controllo da remoto di macchinari, nei videogiochi online o durante videochiamate. Nel LS, invece si può tollerare una latenza più che può variare da qualche secondo a qualche minuto, in base al campo di applicazione e dalle tecnologie utilizzate per la trasmissione. Tendenzialmente il LS viene utilizzato in settori con un bacino di utenti simultanei maggiore rispetto a quelli in real-time streaming, ad esempio concerti ed eventi sportivi, in cui una latenza quasi nulla non è richiesta. Il LS si è oramai confermato come una componente importantissima della comunicazione moderna, trasformando il modo in cui fruiamo e interagiamo con i contenuti video. Piattaforme come YouTube, Twitch, Zoom e Facebook hanno rivoluzionato settori chiave come l'intrattenimento, l'istruzione, l'informazione e lo sport. Un requisito ora fondamentale è la capacità di riuscire a soddisfare le aspettative di QoE degli utenti sempre più numerosi ed esigenti. L'evoluzione delle tecnologie di LS è stata rapida e significativa. Inizialmente, i protocolli tradizionali come *Real-*

Time Transport Protocol (RTP), *Real-Time Messaging Protocol* (RTMP) e *Real-Time Streaming Protocol* (RTSP) erano sufficienti per le esigenze di base, per un pubblico ristretto e con aspettative molto inferiori a quelle attuali. Tuttavia, col passare degli anni la domanda di trasmissioni di qualità elevata e accessibili da una grande varietà di dispositivi ha fatto emergere la necessità di sviluppare soluzioni più avanzate. La QoE, la riduzione della latenza e la scalabilità sono diventate priorità cruciali per i fornitori di servizi di LS. Le sfide tecniche nel LS sono molteplici e complesse. Dalla codifica e decodifica efficiente dei video, alla gestione dinamica del bitrate, fino all'implementazione di architetture di rete avanzate come l'edge computing, ogni aspetto richiede innovazione continua. Questi sviluppi sono essenziali per garantire un'esperienza di streaming fluida e di alta qualità, capace di adattarsi alle condizioni di rete variabili e alle diverse esigenze degli utenti finali.

1.2 Obiettivi

L'obiettivo principale di questa tesi è analizzare e valutare le metodologie più recenti e innovative nel campo del LS prestando attenzione alle tecnologie emergenti e alle strategie di trasmissione in tempo reale. Specificamente, la tesi si propone di:

Presentare i protocolli di trasporto tradizionali e moderni utilizzati nel live streaming, come RTP, RTSP, *HTTP Live Streaming* (HLS), *Dynamic Adaptive Streaming over HTTP* (DASH) e altri ancora. Esaminate le loro caratteristiche analizzeremo i vantaggi, svantaggi e come affrontano le sfide che il LS impone al mondo d'oggi.

Studiare le tecniche di ABR che permettono di adattare dinamicamente la qualità del video in base alle condizioni di rete. Verranno analizzati diversi metodi e strategie per ottimizzare la QoE e minimizzare il buffering.

Presentare tecniche utilizzate per ridurre la latenza nel LS, come l'ABR e il *Chunked Transfer Encoding* (CTE). La tesi esplorerà come queste tecniche vengono integrate nei protocolli contribuendo a migliorare la reattività e la fluidità della trasmissione.

Esaminare applicazioni delle tecnologie e delle metodologie discusse, come delle modifiche o integrazioni a protocolli di trasporto portino particolari vantaggi nelle applicazioni di LS.

Definire il concetto di QoE, di ciò che la influenza e delle metodologie per misurarla e, possibilmente, migliorarla.

Nel capitolo conclusivo, forniremo una panoramica completa dei risultati ottenuti e cercheremo di proporre direzioni future per la ricerca nel campo del LS.

Capitolo 2

Fondamenti del Live Streaming

2.1 Definizione e Concetti Base del live streaming

Come già introdotto nel capitolo precedente lo streaming permette la trasmissione di dati continua da una sorgente ad una destinazione, senza la necessità di scaricare tutti i dati prima di poterne fruire. Ciò è reso possibile dall'utilizzo di tecniche di codifica e protocolli che permettono il trasferimento dei dati dividendoli più porzioni trasmessi in sequenza. Il LS è una applicazione dello streaming dati nel campo del multimedia che mira alla trasmissione in tempo reale di contenuti audio e/o video. Nel caso dello LS di contenuti audio/video la trasmissione di dati contiene sia un flusso audio che un flusso video, entrambi i flussi devono essere gestiti contemporaneamente e con una corretta sincronizzazione tra fotogrammi e traccia audio. Per la trasmissione delle tracce audio video in un unico stream di dati, utile per ridurre il costo finale della trasmissione, si utilizza la tecnica del *multiplexing*, ossia l'inviare più stream di dati diversi sotto forma di un singolo stream [2]. Nel multiplexing vengono presi i diversi stream di dati provenienti dai processi di codifica audio e video, vengono combinati poi in un unico stream e inviato agli utenti, una volta ricevuto dagli utenti avviene il processo di demultiplexing dove lo stream viene diviso nei singoli flussi di dati audio e video, per poi essere decodificati e utilizzati[2]. Svolge inoltre un ruolo cruciale nel mantenere le tracce audio e video sincronizzate. Durante la codifica sia i flussi audio che quelli video vengono compressi e segmentati, ogni segmento è contraddistinto da un *timestamp*, un marcatore temporale che indica il momento esatto in cui quel determinato segmento deve essere riprodotto. Questi marcatori sono essenziali per la sincronizzazione perché durante il multiplexing i flussi audio e video vengono combinati in un unico flusso organizzando i pacchetti in base al loro timestamp. Dal lato utente la procedura di demultiplexing divide il flusso unico nei vari stream audio e video originali e, utilizzando i marcatori, sincronizza le tracce audio e video.

Le componenti fondamentali di un sistema di LS includono:

Codifica: la codifica è il processo di preparazione del video per l'output, in cui il gli stream audio e video vengono codificati da un *encoder* per soddisfare i formati e le specifiche corretti per la registrazione e la riproduzione attraverso l'uso del software di codifica video. Al seguito della codifica avviene il processo di multiplexing con il relativo inserimento dei marcatori temporali e infine la trasmissione verso il server.

Server: punto centrale che riceve il contenuto codificato e lo distribuisce agli utenti finali, spesso usando infrastrutture di rete come un *Content Delivery Network* (CDN). Qua possono avvenire anche processi di transcodifica per migliorare compatibilità e scalabilità.

Decodifica: il processo della decodifica consiste nel demultiplexing dello stream di dati nei singoli flussi audio e video codificati, questi flussi poi vengono decodificati dal *decoder* che li converte in un formato che può essere riprodotto dal dispositivo dell'utente, infine utilizzando i timestamp i flussi audio e video vengono sincronizzati per la riproduzione.

Protocolli di Trasmissione: un protocollo di streaming, noto anche come protocollo di trasmissione, è un metodo standardizzato per trasmettere diversi tipi di contenuti multimediali (solitamente video o audio) attraverso una rete. Nel contesto del LS i protocolli si occupano di gestire aspetti come la suddivisione dei dati in pacchetti, il loro ricomponimento una volta a destinazione, la sequenza in cui vengono trasmessi e la qualità del flusso di dati in base alle specifiche di rete e alle necessità degli utenti.

Un aspetto critico del LS è la QoE. Il documento "A Survey on Quality of Experience of HTTP Adaptive Streaming" [3] esplora come vari parametri, tra cui la latenza e la qualità video, influenzino la QoE e descrive le tecniche per ottimizzare questi fattori. La QoE dipende da diversi elementi, tra cui la stabilità del flusso, la risoluzione del video e il tempo di buffering. Tecniche come l'ABR sono fondamentali per adattare la qualità del video alle condizioni di rete variabili, cercando di offrire all'utente un'esperienza il più fluida e di alta qualità

2.2 Storia ed Evoluzione

Il LS ha subito una significativa evoluzione dai suoi inizi, trasformandosi da una tecnologia di nicchia a un elemento centrale della distribuzione di contenuti multimediali.

Anni '90: Gli Inizi

Con l'avvento di Internet i primi tentativi di streaming erano limitati dalla scarsa velocità di connessione e dalla limitata capacità di elaborazione dei computer. Nel 1995 la società Internet RealNetworks ha lanciato uno dei primi software di streaming, RealPlayer. Nello stesso anno ha ospitato il primo LS pubblico: una partita di baseball tra i New York Yankees e i Seattle Mariners. Vista la bassa qualità e il buffering frequente, la trasmissione è valsa come prototipo funzionale della tecnologia, per poter essere commercializzata e adottata su ampia scala avrebbe avuto bisogno di più successo. Due anni dopo, nel 1997 RealNetwork lanciò RealVideo, uno dei primi programmi commercializzati per il LS, tuttavia non riscosse il successo sperato. Prima che il LS facesse davvero notizia e scalpore passarono anni, quando nel novembre del 1999 si è tenuto il primo webcast presidenziale della storia. Bill Clinton, all'epoca Presidente degli Stati Uniti, ha preso parte ad una discussione online, durante la quale il Presidente e i partecipanti hanno affrontato una serie di questioni delicate, tra cui Medicare e controllo delle armi, tramite domande inviate da oltre 50.000 utenti online collegati alla chat.

Anni 2000: Prime Innovazioni

Con l'aumento della velocità di Internet e la diffusione della banda larga, il LS ha iniziato a guadagnare popolarità. Nel 2002, l'adozione del protocollo RTMP[4] di Adobe Flash ha migliorato notevolmente la qualità dello streaming video. Piattaforme come YouTube, fondata nel 2005, hanno reso il caricamento e la condivisione di video accessibili a tutti, gettando le basi per lo streaming live, infatti nel 2008 lo stesso YouTube ha ospitato il primo evento dal vivo, accessibile a chiunque nel mondo avesse un computer ed una connessione ad Internet. YouTube non è mai, però, diventata una piattaforma specializzata in LS, per anni è rimasto solo un evento occasionale. Col senno di poi la lentezza da parte di YouTube nel proporre più streaming live nel mercato ha lasciato spazio e tempo per altre aziende di approfittare di questa fetta di mercato inesplorato e poco sfruttato.

Anni 2010: Crescita Esponenziale

Nel 2011 infatti nasce Justin.tv(successivamente diventerà Twitch.tv), una piattaforma di intrattenimento interamente dedicata al LS; ciò ha mostrato come il LS non sia riservato ad eventi musicali, sportivi o mediatici, ma anche ai videogiochi, fonte di intrattenimento per oggigiorno miliardi di persone [5]. Inoltre, l'introduzione di protocolli come HLS da parte di Apple nel 2009 e DASH ha migliorato la qualità e l'affidabilità dello streaming su vari dispositivi, rendendolo accessibile a un grande numero di utenti. Il successo di Twitch ha fatto capire al mercato che incredibile fonte di introito fosse il LS, infatti nel 2013 YouTube ha dato la possibilità a qualsiasi utente registrato e con più di 100 iscritti si poter avviare trasmissioni in LS sulla piattaforma. Anche altri social network non potevano mancare questa opportunità e negli anni successivi Facebook, Instragram, Twitter hanno tutti lanciato i loro servizi di streaming dal vivo, rendendo il LS una componente essenziale ed integrante del mondo social di oggi.

Ultimi Anni: Modernizzazione e Innovazione

Negli ultimi anni, il LS ha continuato a evolversi con l'introduzione di tecnologie avanzate come *Quick UDP Internet Connections* (QUIC) di Google, progettato per ridurre la latenza e migliorare la sicurezza delle connessioni, inoltre si ha avuto anche l'avvento del 5G, che promette di rivoluzionare ulteriormente il settore, offrendo velocità di connessione più elevate e minore latenza. La pandemia di COVID-19 ha accelerato ulteriormente l'adozione del LS, con un aumento significativo del traffico internet dedicato a eventi in diretta, concerti virtuali, didattica a distanza e telelavoro[6]. Infatti nel 2018, circa il 23% del traffico internet globale era attribuibile al LS. Questo valore è aumentato significativamente durante la pandemia di COVID-19, quando il traffico internet globale è aumentato del 40% e una grande parte di questo aumento era attribuibile alle trasmissioni live. Quattro anni anni fa secondo il rapporto Sandvine[7], il video rappresentava il 60% del traffico internet globale, con una parte sostanziale proveniente dal LS. Da allora in soli due anni, nel 2022, il LS è arrivato a rappresentare il 23% del tempo di visione globale e il 17% di tutto il traffico internet globale. Piattaforme come Twitch e YouTube Live hanno continuato a crescere, con Twitch che ha raggiunto 2,5 milioni di spettatori contemporanei in media e più di 35 milioni di visite sulla piattaforma al giorno [8]. L'anno successivo la quota di mercato delle principali piattaforme di LS, come YouTube Live Gaming e Twitch, è cresciuta rispettivamente del 2% e del 4%. Il mercato globale del LS è passato da 1,24 miliardi di dollari nel 2022 a 1,49 miliardi di dollari nel 2023. Dagli umili inizi negli anni '90 ai giganteschi progressi tecnologici dei giorni nostri, il LS è

diventato una componente essenziale della nostra vita digitale. Il continuo sviluppo di nuove tecnologie porta all'aumento della domanda di contenuti in tempo reale, grazie questo ciclo continuo di stimoli e sfide il futuro del LS si prospetta pieno di ulteriori innovazioni e strumenti per il miglioramento della qualità del servizio, dei contenuti proposti e della QoE.

Capitolo 3

Tecnologie Tradizionali e Moderne

I protocolli tradizionali per il LS hanno gettato le fondamenta per la trasmissione di contenuti video in tempo reale, alcuni sono stati utilizzati come base per sviluppare altri protocolli, più moderni ed ottimizzati per poter offrire una maggior QoE, scalabilità, ridurre la latenza e un utilizzo più efficace delle moderne infrastrutture dati.

Prima di poter parlare dei protocolli diamo qualche definizione ai concetti e alle architetture di rete da loro utilizzati.

3.1 Concetti e Architetture di Rete Avanzate

Le architetture di rete avanzate sono cruciali per migliorare la scalabilità, l'affidabilità e la qualità del LS. Le CDN, l'EC e il MEC sono alcune delle soluzioni principali adottate per affrontare le sfide del LS.

3.1.1 Content Delivery Network (CDN)

Una Content Delivery Network, o Content Distribution Network, è una rete geografica distribuita di server proxy e data center progettata per garantire alta disponibilità e prestazioni elevate attraverso la distribuzione spaziale dei servizi in relazione agli utenti finali. Nate alla fine degli anni '90, per alleviare i colli di bottiglia delle prestazioni di una rete Internet critica per persone e aziende, servono ora una grande parte dei contenuti in rete, tra cui pagine web (testi, grafica e script), contenuti scaricabili (file multimediali, software, documenti), applicazioni (e-commerce, portali web), media in LS, on-demand e contenuti delle piattaforme social [9]. Una CDN è quindi una infrastruttura di locazioni fisiche chiamate *Point of Presence* (PoP), anche detti nodi della rete CDN e interconnesse tra loro. Distribuiti strategicamente in varie località geografiche per ridurre la distanza fisica tra il contenuto e l'utente finale, i PoP mirano a ridurre la latenza e migliorare i tempi di risposta. Ogni PoP contiene diversi *edge server*, ossia i server che effettivamente memorizzano nella cache i contenuti richiesti più frequentemente. Quando un utente richiede un contenuto, la richiesta viene indirizzata verso il nodo ottimale. In base al tipo di ottimizzazione dell'instradamento della ricerca l'*Internet Service Provider* (ISP) o anche il provider del CDN può decidere se dare priorità l'efficienza della consegna o all'efficienza economica. Nel primo

caso si sceglie quindi il PoP più vicino, con il minor numero di server da attraversare per raggiungerlo o quello con più risorse di rete disponibili in quel momento. Nel caso dell'efficienza economica si tende a preferire PoP il cui utilizzo costa meno in termini monetari. Naturalmente in una CDN progettata in modo efficiente le due priorità coincidono e vengono soddisfatte al tempo stesso, instradando quindi la richiesta verso il PoP più vicino all'utente che, in linea teorica, quasi sempre porta vantaggi in termini di prestazioni e costi[10].

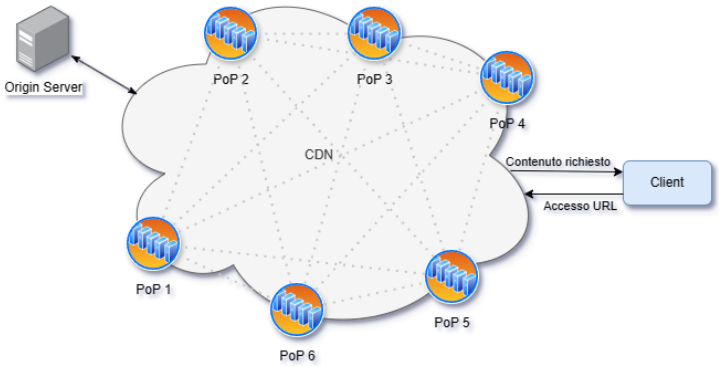


Figura 3.1: Esempio funzionamento di una CDN

Il diagramma 3.1 rappresenta la struttura e il funzionamento di una possibile CDN. Osserviamone il funzionamento:

Struttura della CDN

- **Origin Server**: Il server che contiene la versione originale del contenuto. Questo server può essere situato in qualsiasi parte del mondo e non è necessariamente vicino agli utenti finali.
- **CDN Provider Network**: La rete gestita dal CDN provider, che connette vari PoP distribuiti geograficamente.
- **Points of Presence**: Località geografiche distribuite che contengono i server periferici, o edge servers. Ogni PoP è strategicamente posizionato vicino agli utenti finali per migliorare la velocità di accesso ai contenuti.
- **Edge Servers**: Server all'interno dei PoP che memorizzano in cache i contenuti più richiesti dagli utenti. Questi server sono responsabili di servire il contenuto in memoria direttamente agli utenti finali.

Flusso delle Richieste

- **Accesso URL dal Client**: Un client (utente finale) fa una richiesta di contenuto tramite URL.
- **Contenuto Richiesto**: La richiesta di contenuto viene inviata al PoP più vicino al client.

Gestione delle Richieste: Cache Hit e Cache Miss

Cache Hit

- Se il contenuto richiesto è già memorizzato nell'edge server del PoP più vicino, si verifica un "cache hit".
- L'edge server fornisce il contenuto al client senza dover contattare l'origin server. Riducendo significativamente la latenza e migliorando i tempi di risposta. Anche se la CDN si fa carico di gran parte del traffico il server di origine rimane responsabile per i contenuti dinamici che non possono essere memorizzati in cache o che devono essere aggiornati frequentemente in caso di richiesta di tali contenuti verrà fatta una richiesta direttamente all'origin server.

Cache Miss

- Quando, invece, il contenuto richiesto non è presente nell'edge server, si verifica una "cache miss".
- In tal caso, l'edge server richiede il contenuto all'origin server attraverso la rete CDN.
- Una volta ricevuto il contenuto dall'origin server lo consegna al client. Inoltre utilizzando algoritmi per rilevare la quantità di richieste di utenti dello stesso contenuto, l'edge server decide se è efficiente memorizzare il contenuto nella cache per future richieste o se richiederlo nuovamente all'origin server. Nel caso di memorizzazione nella cache le future richieste dello stesso contenuto da parte di altri utenti che si connettono attraverso lo stesso PoP saranno servite dalla cache dell'edge server, risultando in un cache hit.

Va sottolineato, però, che sebbene esista uno standard di funzionamento generico per le CDN, algoritmi come quelli per la decisione di quale PoP assegnare alla richiesta di un client o per decidere se mantenere un contenuto nella cache o meno e per quanto, sono tutti algoritmi proprietari e il loro funzionamento è a discrezione del gestore del servizio. Stesso discorso per gli algoritmi che vengono utilizzati per la gestione di distribuzione di contenuti all'interno di una CDN, se condividere certi contenuti o meno all'interno dell'intero network in modo che ogni PoP abbia nella cache lo stesso contenuto[11].
Il LS può trarre beneficio dalle CDN grazie alle sue ottimizzazioni e alle tecniche messe in campo per ridurre la latenza. Ad esempio la distribuzione geografica dei PoP e la gestione efficiente dei flussi di rete interni alle CDN possono ottimizzare la ricezione del contenuto in LS, rendendola preferibile alla connessione diretta al server di origine. Se ogni utente dovesse collegarsi direttamente al server di origine, aumenterebbe enormemente il carico sul di esso. Questo non solo richiederebbe al server di inviare simultaneamente dati a tutti gli utenti connessi in tutto il mondo, ma causerebbe anche un notevole aumento della latenza. Il server diventerebbe rapidamente un collo di bottiglia, incapace di gestire il volume di traffico, portando a ritardi, buffering e una QoE complessivamente scadente. Inoltre piattaforme di LS che utilizzano protocolli di trasporto come HLS e DASH(o comunque basati sulla segmentazione del flusso) possono trarre grande vantaggio da una CDN, infatti i segmenti prodotti possono essere facilmente memorizzati nella cache e distribuiti dai PoP non appena resi disponibili, permettendo così agli utenti di iniziare a guardare il LS quasi immediatamente, mentre i segmenti successivi vengono ancora trasferiti.

3.1.2 Edge Computing

L'*Edge Computing* (EC) è un paradigma di calcolo distribuito che coinvolge l'elaborazione e l'analisi dei dati alla fonte o vicino ad essa, piuttosto che inviarli a un data center centralizzato o ad un servizio cloud per l'elaborazione. Il termine "edge" si riferisce al margine di una rete, dove i dati vengono generati, raccolti e analizzati in tempo reale. L'obiettivo principale dell'edge computing è ridurre la latenza e i requisiti di larghezza di banda associati alla trasmissione dei dati verso una posizione centralizzata per l'elaborazione. In questo modo, elaborando i dati al margine della rete, le aziende possono migliorare i tempi di risposta, ridurre la congestione della rete e migliorare le prestazioni complessive delle loro applicazioni.

L'edge computing comporta il dispiegamento di dispositivi edge, come sensori, gateway(nodo di rete, solitamente hardware, che permette la comunicazione tra due reti diverse) e altri tipi di dispositivi di calcolo, al margine della rete. Questi dispositivi sono responsabili della raccolta dei dati, della loro elaborazione in tempo reale e dell'invio ai servizi cloud solo nei casi di dati rilevanti per ulteriori analisi o archiviazione.

L'utilizzo dell'edge computing risulta particolarmente utile in scenari dove la bassa latenza e l'elevata affidabilità sono critiche, come nell'automazione industriale, nei veicoli autonomi e nello streaming live o real-time. Consente, inoltre, alle aziende di elaborare i dati in ambienti dove la connettività di rete è limitata o inaffidabile, come in località remote o su dispositivi mobili. Si tratta, quindi, di uno sviluppo importante nel campo del calcolo distribuito, consentendo alle organizzazioni di migliorare le prestazioni, l'affidabilità e la scalabilità delle loro applicazioni elaborando i dati al margine della rete.

La grande diffusione di dispositivi connessi ad Internet ai margini della rete ha portato un grande aumento all'ammontare dei dati che devono essere calcolati nei data center, portando presto a saturazione le disponibilità di larghezza di banda della rete non potendo più così garantire velocità di trasferimento e tempi di risposta adeguati, requisito significativo per alcune applicazioni. L'obiettivo dell'edge computing è trasferire, per quanto possibile, l'elaborazione lontano dai data center e verso il margine della rete, dove i dispositivi, gli smartphone o i gateway di rete possono eseguire l'elaborazione e fornire servizi per conto del cloud.

Un dispositivo edge è un endpoint della rete che consente ad un data center di interagire con il mondo reale. Questi dispositivi comunicano o raccolgono informazioni. Spaziano dai semplici sensori ai complessi sistemi industriali: possono essere piccoli come scanner e smartphone, termostati intelligenti, campanelli intelligenti o anche grandi come veicoli a guida autonoma, robot industriali e macchinari da lavoro automatizzati. Il cloud computing e l'*Internet of Things* (IoT)(rete di dispositivi fisici contenenti sensori e software che rendono possibile la raccolta e la trasmissione di dati) hanno aumentato il ruolo dei dispositivi ai bordi della rete, richiedendo maggiore capacità di calcolo alla rete edge[12][13][14].

Come rappresentato nella figura 3.2 l'infrastruttura edge segue uno schema piramidale.

- Alla base sta la rete edge composta da i miliardi di dispositivi dell'IoT, che raccolgono, trasmettono e, in alcuni casi, elaborano dati. I dispositivi in questo livello generano un'enorme mole di dati che viene trasferita, solitamente attraverso reti LAN/WAN a degli edge center.

- Gli edge center sono dei nodi della rete edge, composti da server generici che permetto una ristretta elaborazione dei dati. Permettendo una momentanea memorizzazione dei dati nella cache e una piccola quantità di elaborazione si occupano di gestire applicazioni in tempo reale dove non è necessaria la comunicazione con il cloud o con il server di origine.
- Nel caso in cui, invece, le informazioni richiedano ulteriori elaborazioni o uno spazio di memoria elevato, vengono instradate verso i data center cloud attraverso gli edge gateway dove verranno elaborate e, eventualmente, trasferite nuovamente al dispositivo edge finale.

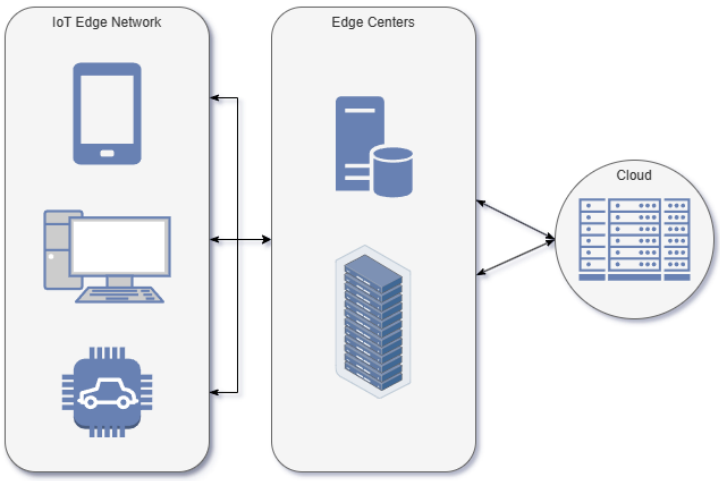


Figura 3.2: Infrastruttura Edge

Come per le CDN, anche se per l'edge computing esiste uno standard di funzionamento generico, starà al gestore del servizio e ai produttori di hardware e software specializzati il compito di decidere il modo in cui trattare i dati prodotti, quanti di questi elaborarli il più vicino possibile e quanti invece affidare agli edge nodes o ai datacenter cloud o proprietari. Nell'ambito del LS e del real-time streaming l'utilizzo dell'edge computing può portare molti benefici; infatti anziché indirizzare tutta l'elaborazione dei dati e gestire tutto il traffico in arrivo ed in uscita generato dagli utenti, si può delegare parte dell'elaborazione dei dati e della diffusione dei contenuti al margine della rete. In questo modo avvicinando l'elaborazione e l'archiviazione del flusso dati live all'utente si ridurrà significativamente la latenza. Piattaforme di LS e creatori di contenuti adoperano già strategie di edge computing per ridurre la latenza, ad esempio Twitch.tv delega la codifica della trasmissione live al creatore di contenuti che sta trasmettendo. Sarà il creatore quindi ad occupare di codificare il flusso dello stream, in questo modo Twitch non dovrà farsi carico anche del processo di codifica, particolarmente richiedente in fatto di risorse computative. Inoltre alcuni creatori di contenuti applicano, nel loro piccolo, il concetto di edge computing. perché oltre ad avere il computer principale utilizzano un secondo computer che, sfruttando una scheda di acquisizione, riceve in input il flusso audio/video del computer principale e lo codifica; in questo modo si solleva il computer principale dal carico di lavoro che è la codifica e la trasmissione, permettendo una codifica a bitrate più alta senza inficiare le prestazioni del computer principale che porta sia ad una riduzione della latenza sia ad un potenziale aumento generale della QoE(aumentano la capacità di calcolo si è meno soggetti a congestione e rallentamenti nella codifica che possono portare a buffering o aumenti di latenza).

3.1.3 Multi-Access Edge Computing (MEC)

Il *Multi-access Edge Computing* (MEC) è uno standard che offre agli sviluppatori di applicazioni e ai fornitori di contenuti capacità di calcolo e archiviazione tipici del cloud computing e un ambiente di servizio sul bordo(edge) della rete qualsiasi essa sia (mobile, WiFi, cablata)[15]. Mira a ridurre la latenza, aumentare la larghezza di banda disponibile e a fornire servizi basati sulla posizione, il che è particolarmente utile per applicazioni di streaming video in tempo reale. I motivi per cui ci sia bisogno dell'edge computing e del MEC in particolare sono molti e il cardine centrale è quasi sempre perché per certe applicazioni i servizi tradizionali di cloud hosting non bastano e non funzionano abbastanza bene[16]. I motivi per cui ciò non è sufficiente sono molteplici; tra i più comuni e diffusi sono:

- L'applicazione richiede una bassa latenza per funzionare correttamente e, pertanto, non è in grado di tollerare la latenza tipica degli ambienti cloud.
- L'applicazione genera una grande mole di dati e l'elaborazione, nonché talvolta anche l'archiviazione, nel cloud è costosa; di conseguenza, è necessaria un'alternativa più economica.
- Necessità di mantenere i dati all'interno della rete aziendale nel caso di dati sensibili o di valore economico.

Quindi il MEC rende possibile il creare un ambiente come quello cloud però sul bordo della rete anziché in luoghi centralizzati e spesso distanti dall'utente finale. Al momento due aziende si stanno occupando[17] di sviluppare questo standard:

- ETSI ISG MEC[18]: gruppo di standard creato esplicitamente per definire la tecnologia di edge computing dal punto di vista informatico senza tenere conto del tipo di accesso. Specifica l'insieme di tecnologie cloud che devono essere implementate dai fornitori di MEC e dai fornitori di applicazioni o contenuti per realizzare piattaforme e applicazioni MEC.
- 3GPPP: standard solitamente incentrati sulla rete mobile, ora dedito anche all'integrazione dell'edge computing nel panorama di sistemi 5G, definendo modi e supporti.

Questi due standard non si ostacolano a vicenda poiché ognuno di occupa di una parte dello standard diversa senza sovrapposizioni, nel campo del 5G sono entrambi al lavoro per fornire uno standard unito per l'applicazione dell'edge computing. Il Multi-Access Edge Computing rappresenta un importante sviluppo nel campo del calcolo distribuito, offrendo miglioramenti significativi in termini di latenza, efficienza e larghezza di banda per molti settori tra cui l'intrattenimento, il settore automobilistico(in particolare le auto a guida autonoma), la medicina e le operazioni industriali. Integrando risorse di calcolo e archiviazione ai margini della rete, MEC consente alle aziende di fornire esperienze di alta qualità agli utenti finali, rispondendo alle crescenti esigenze moderne.

3.2 Protocolli Tradizionali

Real-Time Transport Protocol (RTP), *Real-Time Messaging Protocol* (RTMP) e *Real-Time Streaming Protocol* (RTSP) sono tre principali protocolli di streaming utilizzati per la trasmissione di contenuti multimediali in tempo reale su Internet. Nonostante la crescente popolarità di tecnologie più recenti

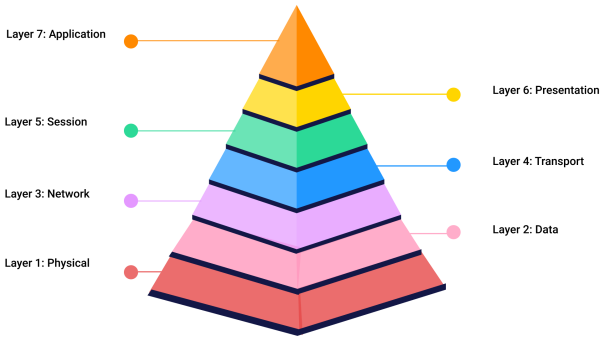


Figura 3.3: Modello OSI

come *HTTP Live Streaming* (HLS) e *Dynamic Adaptive Streaming over HTTP* (DASH), RTP, RTMP e RTSP continuano ad essere ampiamente utilizzati in vari contesti.

3.2.1 Real-Time Transport Protocol (RTP)

Standardizzato dall'*Internet Engineering Task Force* (IETF)[19], il RTP è un protocollo che opera al livello di sessione (session-level protocol) progettato per la trasmissione end-to-end di dati in tempo reale come audio, video o dati di simulazioni. Solitamente utilizza l'*User Datagram Protocol* (UDP) per il trasporto, incapsulando il pacchetto RTP in uno UDP; così aggiungendo alla bassa latenza, alla semplicità e al supporto del multicasting di UDP capacità come il rilevare la perdita di pacchetti e la sincronizzazione temporale dei dati[20]. Rappresenta un tassello importante per applicazioni che richiedono la trasmissione di dati in tempo reale con attenzione alla sincronizzazione e alla gestione efficace della sequenza dei pacchetti di dati[21]. Sebbene lo sviluppo iniziale fosse per supportare audio-e-video conferenze è stata sviluppata con l'intenzione di renderla il più flessibile ed adattabile a diversi scenari d'uso[22]. RTP può essere utilizzata sia in ambienti unicast(punto-a-punto) che multicast(uno-a-tanti o tanti-a-tanti) e può scalare facilmente da uno a migliaia di endpoint(dispositivo connesso ad una rete). Nell'RFC 3550[19] vengono inoltre definiti i concetti di *mixer* e *translator*. Il primo, quando riceve flussi di dati da una o più fonti, gli unisce per creare un nuovo flusso dati combinato. Quindi il mixer genera dei timestamp e sincronizza i flussi. Il translator, invece, inoltra i pacchetti RTP lasciando intaccata la sincronizzazione; può anche cambiare la codifica del flusso e replicare i pacchetti per la trasmissione da multicast a unicast (il flusso multicast può risultare più affidabile verso client wireless)[23][22].

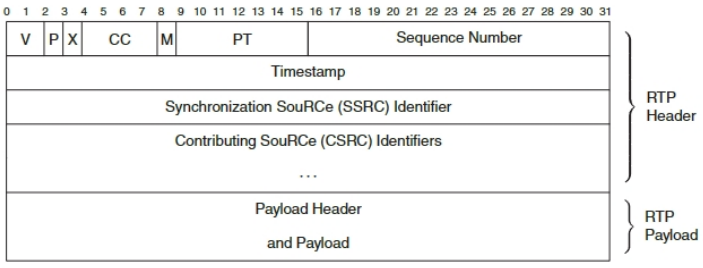


Figura 3.4: Diagramma pacchetto RTP

La figura 3.4 mostra il layout di un pacchetto RTP, sebbene ogni parte di esso svolge un ruolo importante, ci concentriamo sul *Timestamp*, il *Sequence Number* e il *Payload Type*, perché rappresentano i componenti

essenziali per gestire la sincronizzazione, l'ordine corretto e l'interpretazione dei dati multimediali.

- **Timestamp:** campo lungo 32 bit, segna l'istante di codifica del primo ottetto di byte del payload(carico di dati contenuti nel pacchetto). Utilizzando un orologio di riferimento del mittente, il timestamp permette al ricevitore di riprodurre i dati con la corretta temporizzazione[19][24].
- **Sequence Number:** serve per identificare i pacchetti e fornire al ricevitore una indicazione per capire se dei pacchetti sono stati persi[24]. Il campo contiene un numero di sequenza di 16 bit di partenza casuale(non 0 per motivi di sicurezza) che viene incrementato di uno per ogni pacchetto trasmesso. Se il numero di pacchetti persi supera 100, il flusso viene interrotto e riavviato[23]. L'uso primario del sequence number è per poter rilevare la perdita di pacchetti consecutivi, serve inoltre per poter ricostruire l'ordine in cui i pacchetti sono stati inviati[22].
- **Payload Type:** identifica il tipo di media trasportato dal pacchetto, è lungo 7 bit e determina come i dati devono essere interpretati dal ricevitore. Ogni valore del payload type corrisponde a un codec specifico o a un formato di dati, come audio compresso, video compresso o dati di testo[22].

3.2.2 Real-Time Messaging Protocol (RTMP)

RTMP è stato inizialmente sviluppato da Macromedia e ora è di proprietà di Adobe. In principio era utilizzato per facilitare lo streaming di contenuti audio, video e dati da un server a un client che utilizzava un plug-in del browser(Flash Player). Inizialmente veniva utilizzato per videoconferenze, durante le quali lo stesso client accedeva al microfono e alla videocamera dell'utente per poi utilizzare RTMP per inviare e ricevere flussi di dati. Dopo l'acquisto di Macromedia da parte di Adobe è stato rilasciato un documento tecnico relativo ad una versione pubblica di RTMP [25]. RTMP è un protocollo al livello applicazione (application-level protocol) basato sul *Transmission Control Protocol* (TCP), di cui sfrutta l'affidabilità, la capacità di ordinamento dei pacchetti e di mantenere una connessione stabile a bassa latenza; eredita inoltre la stessa tecnica a tre fasi: *handshake*, *connection* e *stream* schematizzata in 3.5.

Durante l'*handshake* avviene lo scambio di 3 pacchetti di dati tra il client e il server per stabilire la

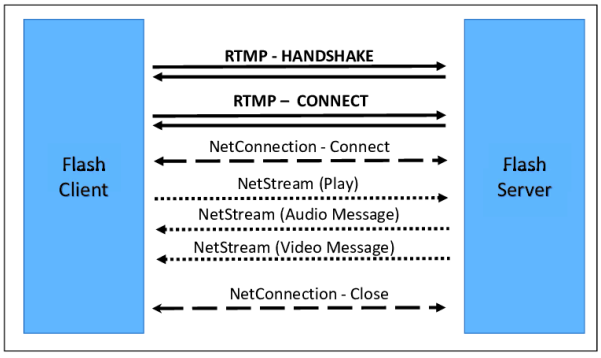


Figura 3.5: Sessione di streaming RTMP[26]

versione di RTMP usata e sincronizzare i timestamp. Con la *connection* il client invia al server una richiesta di connessione sotto forma di *Action Message Format* (AMF) specificando dettagli come l'URL e il codec dei dati, la struttura di AMF facilita l'adattamento del flusso alle condizioni di rete variabili. Infine nella *stream(o NetStream)* viene reso possibile l'utilizzo di comandi per la gestione della trasmissione dei dati [25]. Per mantenere flussi di dati costanti e trasmettere più informazioni possibili mantenendo

un basso overhead(utilizzo di risorse dal parte del protocollo, non contando quelle utilizzate dai dati trasportati), RTMP divide i flussi in segmenti di dimensione dinamica, ossia di dimensioni variabili decise durante la comunicazione tra server e client. Come mostrato in 3.6 ogni pacchetto è diviso in

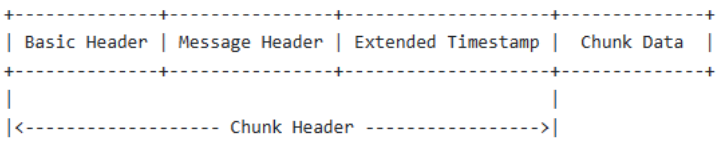


Figura 3.6: Diagramma pacchetto RTMP[25]

header(a sua volta diviso in tre parti) e data:

- **Basic Header:**Determina come devono essere interpretate le informazioni dell'header e specifica quale stream ID(dato identificativo univoco del flusso) è associato al chunk.
- **Message Header:** contiene informazioni riguardanti il timestamp, utilizzato per sincronizzare i dati, la lunghezza del messaggio del frammento, identifica il tipo di message(dato codificato in AMF) trasportato (audio, video, ecc.) e lo stream ID. Ci sono vari formati possibili di questo header nei quali cambia la quantità di informazioni presenti al suo interno, è da utilizzare preferibilmente quella più compatta possibile per l'utilizzo ricercato[25].
- **Extended Timestamp(opzionale):** contiene un timestamp esteso utilizzato solo se il timestamp supera il valore massimo rappresentabile nei 3 byte standard del timestamp.
- **Chunk Data:** il payload, ossia la porzione di dati effettivi del flusso.

RTMP offre un LS veloce e affidabile con una latenza minima e buone capacità di scalabilità, grazie anche al supporto per il multicasting e alla definizione di canali virtuali multipli con cui può effettuare il multiplexing di audio, video e dati trasmessi nella stessa connessione TCP[25]. Tuttavia, RTMP presenta alcuni svantaggi, come la dipendenza da Adobe Flash Player, tecnologia dismessa da anni, oltre che alla necessità di un server dedicato[27]. L'utilizzo di TCP, però, oltre alla stabilità e all'ordine di consegna dei pacchetti, porta con se anche latenza aggiuntiva dovuta dal controllo della connessione, del flusso e della eventuale ritrasmissione dei pacchetti persi. Questo può essere meno efficiente per applicazioni che richiedono latenze estremamente basse.

3.2.3 Real-Time Streaming Protocol (RTSP)

RTSP[28] è un protocollo a livello applicazione che da la possibilità al client di controllare in tempo reale la trasmissione di multipli flussi multimediali inviati da un server, flussi che possono provenire sia da dati presenti e salvati nei server, sia da fonti live. Tali controlli includono comandi come *start, stop, pause, repositioning playback to future or past point of time, fast forwarding, rewinding playback* e altri, RTSP si comporta come un "telecomando" di rete remoto per server multimediali[28][29]. RTSP è un protocollo senza connessione che controlla il flusso multimediale tra client e server[20]. Non esistendo una connessione RTSP, il server collega la sessione ad un identificativo univoco, mentre come protocollo di trasporto utilizza solitamente RTP appoggiandosi ad UDP o TCP in base alle necessità di utilizzo; UDP nel caso sia importante la velocità della trasmissione dati e una bassa latenza a discapito della possibile

non ricezione di pacchetti, TCP se si preferisce l'affidabilità e il rinvio dei pacchetti persi; fornendo anche la possibilità di scegliere i canali per la consegna dei contenuti multimediali, per esempio TCP, singlecast UDP e multicast UDP [21] [23]. RTSP è stato progettato con in mentre l'idea fornire servizi per i flussi di audio e video nello stesso modo in cui *Hypertext Transfer Protocol* (HTTP) li fornisce per il testo e per la grafica [21][23]. Per questo motivo la sintassi e i comandi sono molto simili a quelli di HTTP, in questo modo molte sue estensioni possono essere integrate facilmente anche in RTSP. Nella figura 3.7 è

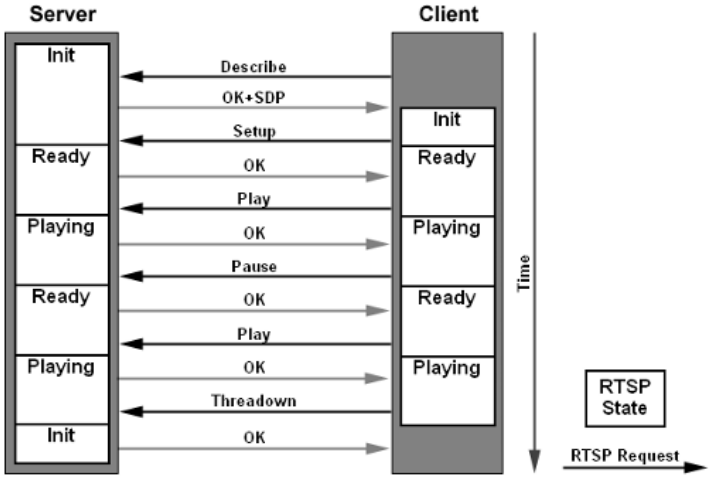


Figura 3.7: Diagramma sessione RTSP[30]

presentato un esempio di una tipica sessione RTSP tra client e server.

- **Describe**: Il client invia un comando `DESCRIBE` al server per ottenere una informazioni su un determinato contenuto multimediale. Questo comando chiede informazioni come la descrizione della sessione, i media disponibili e le relative caratteristiche.
- **OK + SDP**: Il server risponde con un messaggio di conferma `OK` e fornisce la descrizione della sessione tramite *Session Description Protocol* (SDP). Questa risposta include informazioni sui flussi multimediali, codec utilizzati, porte di rete e altre informazioni necessarie per configurare la sessione.
- **Setup**: Il client invia un comando `SETUP` per configurare il flusso di media specifico. Questo comando indica al server di allocare risorse per il flusso multimediale e di stabilire i parametri di trasporto (ad esempio, l'uso di UDP o TCP, le porte, ecc.).
- **OK**: Il server risponde con `OK`, confermando che la configurazione del flusso multimediale è stata completata. A questo punto, la sessione è pronta e il server entra nello stato "Ready".
- **Play**: Il client invia un comando `PLAY` per iniziare la riproduzione del flusso multimediale.
- **OK**: Il server risponde con `OK`, indicando che la riproduzione è stata avviata con successo. Sia il client che il server passano allo stato "Playing".
- **Pause**: Il client invia un comando `PAUSE` per mettere in pausa la riproduzione del flusso multimediale.
- **OK**: Il server risponde con `OK`, confermando che la riproduzione è stata messa in pausa. Client e server tornano allo stato "Ready".

- **Play**: Il client invia nuovamente un comando `PLAY` per riprendere la riproduzione del flusso multimediale.
- **OK**: Il server risponde con `OK`, indicando che la riproduzione è stata ripresa con successo. Come prima, entrambi passano allo stato "Playing".
- **Teardown**: Il client invia un comando `TEARDOWN` per terminare la sessione RTSP. Questo comando indica al server di rilasciare tutte le risorse allocate per il flusso multimediale.
- **OK**: Il server risponde con `OK`, confermando che la sessione è stata terminata. Entrambi il client e il server tornano allo stato "Init" iniziale.

RTSP è stato molto utilizzato come protocollo dopo la sua definizione nel 1998[28] e da allora molti passi avanti sono stati fatti per superare le sue limitazioni e per sfruttare a pieno le sue potenzialità. Per questo motivo nel 2016 sono state pubblicate le specifiche per il RTSP2.0 nell'RFC7826[31], tuttavia anche se basato sull'originale oltre ai meccanismi per la negoziazione della versione tra client e server, non è compatibile con le sue versioni precedenti. Tra le novità introdotte nella nuova versione troviamo il "request pipelining" ossia la possibilità del client di avere più di una richiesta in sospeso e inviarle(in ordine) tramite la stessa connessione[31] potenzialmente velocizzando l'avvio della sessione e riducendo la latenza end-to-end.

3.2.4 HTTP Live Streaming (HLS)/Low-Latency HLS (LL-HLS)

HLS è un protocollo a livello applicazione sviluppato da Apple e presentato nell'RFC8216 con una seconda edizione in fase di lavorazione[32][33], grazie al grande successo dell'iPhone e dell'iPad basati su iOS, ha ricevuto un forte sostegno si è fatto spazio quasi forzatamente nel mercato dei protocolli essendo l'unico supportato nativamente su tali dispositivi [34].

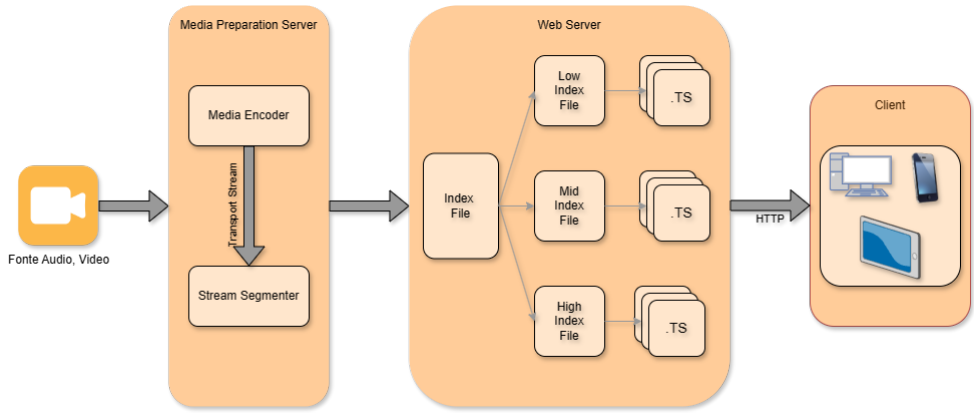


Figura 3.8: Schema funzionamento HLS

Nella figura 3.8 è presentato lo schema di come il flusso di dati viene gestito da HLS

- **Input audio, video:** questi possono provenire da diverse fonti, come videocamere, microfoni o file multimediali pre-registrati.

- **Media Encoder**: Gli input audio e video vengono inviati a un encoder multimediale. L'encoder converte gli input grezzi in un formato digitale compresso, adatto per la trasmissione. Tipicamente, H.264 è utilizzato per il video e AAC per l'audio.
- **Stream Segmenter**: Dopo la codifica, i flussi multimediali compressi passano attraverso un segmentatore di flusso. Il segmentatore divide il flusso continuo in piccoli segmenti di durata prefissata(tra i 6 e i 10 secondi). Questi segmenti sono salvati come file separati con estensione `.ts` (Transport Stream).
- **Index File**: Il segmentatore crea anche uno o più file di indice (playlist) in formato M3U8. Questi file contengono una lista di tutti i segmenti disponibili, con i loro *Uniform Resource Identifier* (URI). La playlist include anche informazioni su ciascun segmento, come la durata, e permette al client di sapere quale segmento riprodurre successivamente.
- **Divisione con ABR**: Si applica l'ABR, quindi il contenuto viene codificato in diverse qualità a bitrate diversi, per adattarsi a diverse larghezze di banda. Ogni qualità ha il proprio file di indice; i segmenti e i relativi file di indice vengono ospitati sul server web.
- **Distribuzione tramite HTTP:**I contenuti (playlist e segmenti) vengono distribuiti attraverso Internet utilizzando il protocollo HTTP. Questo approccio facilita l'integrazione con le infrastrutture di rete esistenti e l'utilizzo di CDN per distribuire i contenuti in modo efficiente a un vasto pubblico.
- **Client:** Il client, che può essere un browser web, un'applicazione mobile o un dispositivo di streaming, scarica la playlist iniziale dal server HTTP. In base alla playlist, il client richiede i segmenti successivi dal server e li riproduce in sequenza, assicurando una riproduzione continua del contenuto multimediale. Il client può adattare la qualità dello streaming in base alle condizioni di rete e alla larghezza di banda disponibile, selezionando versioni diverse dei segmenti (ad esempio, a risoluzioni o bitrate differenti) elencate nella playlist.

Per superare alcune limitazioni di HLS e per migliorare la QoE in applicazioni che richiedono una latenza particolarmente bassa, Apple ha sviluppato una versione del protocollo improntata alla bassa latenza, *Low-Latency HTTP Live Streaming* (LL-HLS)[35]. Mentre HLS utilizza segmenti che arrivano a durare anche 10 secondi, potenzialmente introducendo latenza tra la cattura del contenuto e la visualizzazione dagli utenti, *Low-Latency HTTP Live Streaming* (LL-HLS) suddivide questi segmenti in segmenti ancora più piccoli, anche di pochi centesimi di secondo, riducendo così la latenza rispetto a HLS.
Il preloading delle playlist in LL-HLS è una delle innovazioni chiave progettate per ridurre la latenza complessiva dello streaming. Questo meccanismo permette al client di iniziare a scaricare parti del segmento prima che il segmento completo sia disponibile, migliorando la reattività e riducendo i tempi di attesa[35]. La master playlist (file .M3U8 principale) contiene riferimenti a playlist corrispondenti a vari livelli di qualità (bitrate). Ogni playlist per livello di qualità contiene riferimenti ai segmenti e alle loro parti. Include inoltre informazioni sulle parti imminenti tramite istruzioni di precaricamento (preload hints). Le playlist in LL-HLS includono delle istruzioni chiamate `EXT-X-PRELOAD-HINT`, che informano il client che una parte di segmento sarà disponibile a breve. Queste istruzioni indicano al client di prepararsi a scaricare queste parti appena sono pronte.

Esempio di Preload Hint in una Playlist

```
#EXTM3U
#EXT-X-TARGETDURATION:2
#EXT-X-MEDIA-SEQUENCE:1234
#EXT-X-PART:DURATION=0.333,URI="segmento1234.1.ts"
#EXT-X-PART:DURATION=0.333,URI="segmento1234.2.ts"
#EXT-X-PART:DURATION=0.334,URI="segmento1234.3.ts"
#EXT-X-PRELOAD-HINT:URI="segmento1235.1.ts"
```

Quando il client riceve le parti di segmento già disponibili, inizia a scaricarle e riprodurle immediatamente. Non appena una nuova parte di segmento viene resa disponibile dal server, il client, grazie ai suggerimenti di precaricamento(preload hints), procede al download immediato. Questo meccanismo riduce significativamente il tempo di attesa, consentendo una riproduzione più fluida e continua[35]. Il precaricamento delle parti dei segmenti permette al client di iniziare a scaricare i dati non appena diventano disponibili, si riduce così la latenza complessiva dello streaming. Infatti il client non deve attendere il completamento dell'intero segmento prima di iniziare la riproduzione, in questo modo possiamo ottenere una latenza minima e una buona QoE. Il flusso di dati verso il client, grazie alle preload hints, rimane stabile, abbassando le probabilità di interruzioni e buffering, particolarmente vantaggioso per applicazioni live, dove la continuità del flusso è molto richiesta. Distribuendo il carico di download nel tempo e consentendo ai client di scaricare i dati man mano che diventano disponibili, si ottimizza l'uso della larghezza di banda e si riducono i picchi di traffico facendo uso più efficiente delle risorse di rete. Tuttavia, le playlist in LL-HLS devono essere aggiornate molto più frequentemente rispetto a quelle tradizionali utilizzate da HLS. È richiesta quindi una infrastruttura di server capace di gestire aggiornamenti rapidi e frequenti, mantenendo la reattività necessaria per supportare la bassa latenza. L'uso di segmenti più brevi e delle loro partizioni può rendere più complesso il caching sui CDN. Tuttavia, grazie all'imposizione sul mercato ereditata da HLS, molti CDN moderni supportano LL-HLS e sono ottimizzati per gestire questo tipo di traffico, riuscendo così ad offrire una distribuzione efficiente dei contenuti anche in un contesto di streaming a bassa latenza.

3.2.5 Dynamic Adaptive Streaming over HTTP (DASH)/Low-Latency DASH (LL-DASH)

DASH, spesso indicato come MPEG-DASH, non è effettivamente un protocollo, è bensì uno standard internazionale di formati per lo streaming adattivo di contenuti multimediali su HTTP. Sviluppato dal *Moving Picture Experts Group* (MPEG), DASH consente di fornire video di alta qualità su Internet regolando dinamicamente la qualità del flusso video in base alle condizioni di rete in tempo reale[36] condividendo anche vari aspetti chiave con i più famosi protocolli proprietari[23]:

- Utilizzo di un indice di qualche tipo e file che descrivono gli stream
- Il client può decidere e cambiare il bitrate dello stream

- Per il trasporto viene utilizzato HTTP, facilitando così l'utilizzo delle comuni infrastrutture di rete odierne
- I segmenti di flusso vengono archiviati come file individuali

Come l'HLS, DASH divide il contenuto audio e video in segmenti di corta durata. Questi segmenti sono generalmente di pochi secondi e possono essere codificati a diversi bitrate. I client scaricano questi segmenti in sequenza e li riproducono, iniziando la riproduzione quasi immediatamente dopo aver scaricato il primo segmento[36]. Utilizzando HTTP per il trasporto dei segmenti, può sfruttare le infrastrutture di rete già esistenti, come i CDN. Inoltre lo stream utilizza porte standard quasi sempre aperte e libere, evitando così problemi con il firewall, il quale solitamente blocca protocolli di streaming che utilizzano porte non standard[37]. Oltre al contenuto segmentato viene anche fornito un *Multimedia Presentation Description* (MPD), mostrato in 3.9, un file XML che descrive tre elementi importanti: *Periods*, *Adaptations*, *Representations* e *Segments*. I **Periods** sono intervalli temporali del contenuto multimediale. Ogni

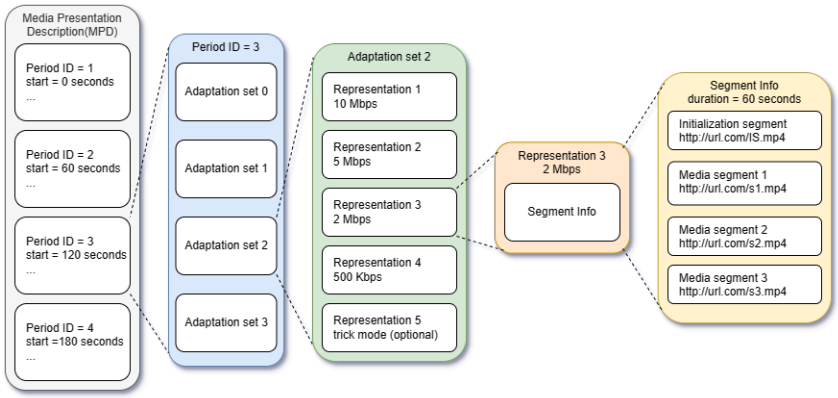


Figura 3.9: Composizione file MPD

MPD può essere suddiviso in uno o più periods, ciascuno con un ID unico e un tempo di inizio specificato. Ogni period contiene uno o più adaptation sets. Gli **Adaptation Sets** sono gruppi di representations alternative dello stesso contenuto multimediale. Ogni set può contenere representations con differenti bitrate, risoluzioni o lingue. Per esempio, un set può contenere representations a diversi bitrate per la componente video, mentre un altro quelle per la componente audio. Le **Representations** sono versioni alternative del contenuto all'interno di un adaptation set. Ogni representation ha un bitrate diverso, permettendo al client di adattarsi alle condizioni di rete variabili. Forniscono così al client opzioni diverse per lo streaming e la riproduzione del contenuto. Opzionalmente si può anche utilizzare anche una representation chiamata *trick mode* che imita il feedback visivo fornito durante le operazioni di avanzamento rapido e riavvolgimento fornite da sistemi analogici come i videoregistratori. I **Segments** sono le unità fondamentali di contenuto all'interno di una representation. Ogni segment ha una durata specifica e può includere segmenti di inizializzazione e i segmenti successivi. L'MPDcontiene gli URL per ciascun segmento, consentendo al client di scaricarli successivamente l'uno dopo l'altro. Per adattare la qualità dello streaming in base alle condizioni di rete del client DASH utilizza l'ABR; utilizzando 3.9 possiamo ipotizzare un suo scenario di applicazione:

1. **Analisi delle Condizioni di Rete**:
    - Il client monitora costantemente la propria larghezza di banda e le condizioni di rete.
    - Utilizza queste informazioni per decidere quale representation scegliere per cercare di minimizzare il buffering e offrire una riproduzione fluida[3][38].
2. **Selezione della Representation**:
    - In base alla larghezza di banda disponibile, il client seleziona la representation appropriata dagli adaptation sets.
    - Ad esempio, con una larghezza di banda elevata, il client potrebbe scegliere una representation con bitrate alto (ad es. 10 Mbps).
    - Se la larghezza di banda diminuisce, il client può passare a una representation con bitrate inferiore (ad es. 2 Mbps).
3. **Download dei Segments**:
    - Il client scarica i segmenti della representation selezionata.
    - Se le condizioni di rete cambiano durante lo streaming, il client può adattarsi scaricando segmenti da una representation diversa.

Essendo indipendente dai codec, può essere utilizzato con vari formati di compressione video e audio, come H.264, H.265, VP9 e AV1 (i più famosi) in questo modo si presta a una vasta gamma di applicazioni, al contrario di HLS, che richiede esclusivamente l'uso di H.264 o H.265 [35][37]. Come nel caso del HLS e del LL-HLS anche DASH ha una sua versione a bassa latenza, il *Low-Latency DASH* (LL-DASH). Esso è un'ulteriore sviluppo dello standard progettato per ridurre la latenza dei contenuti in LS. Esattamente come nel corrispettivo della Apple, si ottiene una riduzione della latenza dividendo i segmenti in ulteriori "parti di segmento" e i file MPD vengono aggiornati e trasmessi in tempo reale. Essenzialmente l'architettura e la segmentazione funzionano allo stesso modo, entrambi trasmettono attraverso HTTP, godendo così dei servizi e delle caratteristiche della rete, entrambi offrono una latenza più bassa (al massimo qualche secondo); tuttavia l'aggiornamento frequente dei file MPD richiede una infrastruttura di server capace di gestire questi aggiornamenti rapidi e frequenti.

3.3 Protocolli Moderni

Le tecnologie moderne hanno superato molte delle limitazioni dei protocolli tradizionali, introducendo miglioramenti significativi nella latenza, affidabilità e scalabilità del LS. Tra questi, *Web Real-Time Communication* (WebRTC) e *Quick UDP Internet Connections* (QUIC) sono particolarmente rilevanti.

3.3.1 Web Real-Time Communication (WebRTC)

Web Real-Time Communication è una tecnologia open source a livello applicazione, sviluppata da IETF e *World Wide Web Consortium* (W3C)[39], consente la comunicazione diretta in tempo reale tra browser

utilizzando il protocollo RTP[38].Il supporto per WebRTC è di default inserito in tutti i browser permettendo così agli sviluppatori di creare applicazioni di comunicazione senza richiedere plugin aggiuntivi. Inizialmente era concepito solo come strumento da utilizzare per le videoconferenze (Microsoft Teams, che è esploso in popolarità durante la pandemia, utilizza WebRTC per le comunicazioni audio e video). Come altri protocolli, tipo HLS e DASH, anche WebRTC supporta l'ABR. Una delle caratteristiche principali di WebRTC è la sua capacità di stabilire una comunicazione diretta tra due dispositivi (P2P). Nell'immagine 3.10 è rappresentato uno schema complessivo di WebRTC, in particolare:

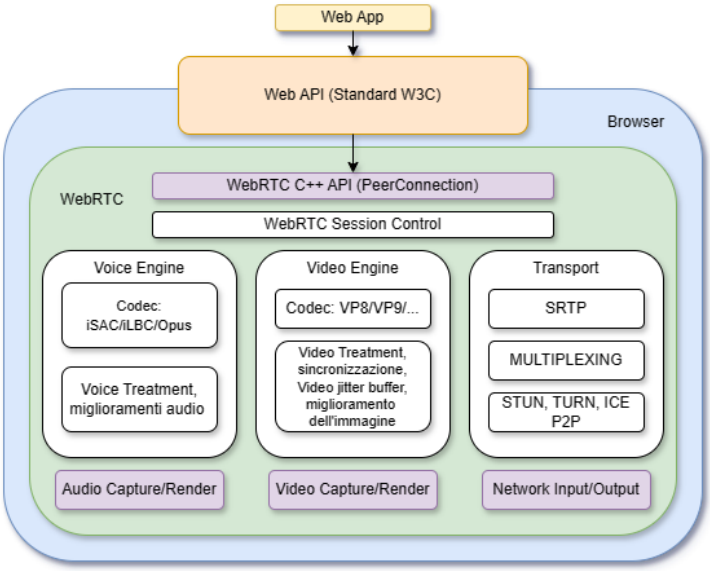


Figura 3.10: Schema WebRTC

- **Web App**: L'applicazione web che utilizza WebRTC per fornire funzionalità di comunicazione in tempo reale.
- **Web API (Standard W3C)**: Le *Application Programming Interface* (API) standardizzate dal W3C[39][40] utilizzate dagli sviluppatori per creare applicazioni web[41]. Le API principali includono `getUserMedia` per l'accesso a microfono e videocamera, `RTCPeerConnection` per la creazione di un canale di comunicazione peer-to-peer, e `RTCDataChannel` per la creazione di un canale dati bidirezionale[23].
- **WebRTC C++ API**: Un livello di API che consente agli sviluppatori di browser di implementare le Web API sopracitate
- **WebRTC Session Control**: Controlla e gestisce la sessione di comunicazione WebRTC.

Voice Engine

- **Codec Audio**: WebRTC supporta vari codec audio come iSAC, iLBC[42], e Opus[43], che sono utilizzati per codificare e decodificare l'audio in tempo reale.
- **Voice Treatment**: Comprende il trattamento del segnale vocale, miglioramenti audio, riduzione del rumore e cancellazione dell'eco.

Video Engine

- **Codec Video**: WebRTC supporta codec video come VP8 e VP9(al momento di stesura della tesi è a livello sperimentale)
- **Video Treatment**: Includono il trattamento del video, la sincronizzazione, il buffer per il jitter (porzione di memoria dove i dati audio/video in arrivo vengono temporaneamente raccolti durante una LS prima di essere decodificati, in questo modo si cerca di evitare instabilità nella riproduzione del flusso dovuta alla ricezione dei pacchetti ad intervalli di tempo non regolari) e il miglioramento dell'immagine.

Transport

- ***Secure Real-Time Transport Protocol* (SRTP)**: Utilizzato per garantire la sicurezza delle trasmissioni in tempo reale, criptando i flussi audio e video.
- **Multiplexing**: Trasmissione di più flussi di dati sotto forma di un singolo flusso.
- **STUN, TURN, ICE, P2P**: Utilizzo di *Interactive Connectivity Establishment* (ICE), uno standard che utilizza i protocolli *Traversal Using Relay around NAT* (TURN) e *Session Traversal Utilities for NAT* (STUN) per stabilire una connessione tra due punti dispositivi connessi alla rete.

Processi di Cattura e Rendering

I browser, a discrezione dei loro sviluppatori, possono prendere il controllo di questi processi.

- **Audio Capture/Render**: Gestisce la cattura dell'audio dal microfono e la riproduzione dell'audio attraverso gli altoparlanti.
- **Video Capture/Render**: Gestisce la cattura del video dalla videocamera e la visualizzazione del video sullo schermo.
- **Network Input/Output**: Gestisce l'ingresso e l'uscita dei dati di rete.

Come già introdotto nella descrizione dei metodi di trasporto dei dati WebRTC utilizza varie tecnologie per offrire comunicazione peer-to-peer in tempo reale tra browser: SDP, ICE e RTP; avendo già parlato di RTP all'inizio del capitolo, ci concentriamo sugli altri due componenti. SDP è protocollo semplice che serve per controllare che entrambi i browser intenzionati a creare una connessione tra loro abbiano dei codec in comune, nel caso questi codec comuni non fossero presenti la connessione non è possibile. La comunicazione di queste stringe contenenti i codec avviene grazie al **Signaling Server** di cui parleremo in seguito. ICE come abbiamo accennato è una tecnica utilizzata per stabilire la connessione tra due browser anche se nascosti dietro al *Network Address Translation* (NAT)(strumento per mappare e collegare multipli indirizzi di rete privati dentro una rete locale ad un solo indirizzo IP pubblico).

Nell'immagine 3.11 il browser A utilizza il server STUN per conoscere il proprio indirizzo IP, il browser B fa la stessa cosa e comunica i propri IP locali e pubblici al browser A usando un Signaling Server; dopodiché il browser A manda dei ping al browser B usando gli IP ricevuti e poi sceglie quello con tempo

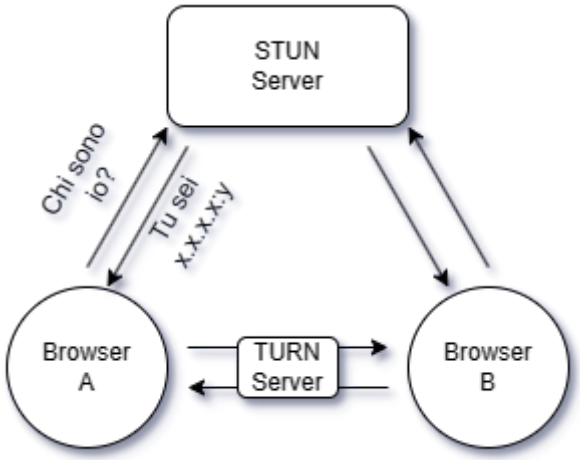


Figura 3.11: Come ICE utilizza TURN e STUN

di risposta e altre metriche migliori, infine il browser B compie lo stesso processo e così si instaura una connessione tra i due browser.

Il server TURN serve nel caso in cui per qualche motivo la connessione tra i due browser usando STUN non è possibile, in tal caso farà da server di inoltro per far comunicare i due browser.

Del concetto di Signaling Server non esiste una definizione o una implementazione standard nella documentazione di WebRTC, quindi ogni sua implementazione ha un suo stile ed è un opera a sé, dal lato applicativo però viene utilizzato per scambiare SDP e indirizzi IP provenienti da ICE tra un client browser e l'altro.

Nonostante i numerosi vantaggi, l'implementazione di WebRTC può presentare alcune sfide. La gestione delle connessioni peer-to-peer, l'attraversamento dei NAT e dei firewall, e la negoziazione dei codec comuni mediante SDP possono essere complessi. Inoltre, sebbene i browser mobili supportino WebRTC, l'esperienza e le prestazioni possono variare rispetto ai browser desktop. L'aumento dell'utilizzo di piattaforme di LS negli ultimi anni, con utenze sempre più esigenti in termini di QoE, ha reso evidente la necessità di combinare le capacità di protocolli come DASH e WebRTC[27]. È stata creata una implementazione di prova che si può trovare presso [44], inoltre Cloudflare all'inizio del 2024 ha annunciato l'inizio del beta testing di WebRTC per offrire ai suoi clienti LS con latenza inferiore al secondo[45].

3.3.2 Quick UDP Internet Connections (QUIC)

QUIC è un protocollo che opera a livello di trasporto sviluppato inizialmente da Google[46] e poi standardizzato dall'IETF[47]. Nato per migliorare le prestazioni dei protocolli di trasporto tradizionali basati su TCP[27], utilizzando il multiplexing di più flussi all'interno di una singola connessione UDP. Ciò significa che più richieste e risposte possono essere inviate in parallelo senza dover attendere il completamento dei flussi precedenti, potenzialmente riducendo la latenza e migliorando l'efficienza della rete. Una particolare capacità di QUIC è quella di ridurre il tempo necessario per stabilire una connessione sicura, infatti QUIC permette una connessione istantanea (0-*Round Trip Time* (RTT))[47][48] con server con i quali si è già collegato in passato. Nel caso di un server nuovo, o nel caso nella cache del client mancassero i parametri necessari, l'handshake avverrà in 1-RTT, evitando comunque i canonici 1.5-RTT previsti dagli handshake delle connessioni TCP, senza contando quelli necessari per *Transport Layer Security* (TLS)

1.2(protocollo crittografico progettato per garantire la sicurezza delle comunicazioni nella rete). Infatti QUIC utilizza la crittografia end-to-end come impostazione predefinita utilizzando TLS 1.3, garantendo che tutte le comunicazioni siano protette. Questo rappresenta un grande miglioramento rispetto a TCP, dove la crittografia deve essere implementata separatamente utilizzando TLS[48]. Il diagramma in 3.12

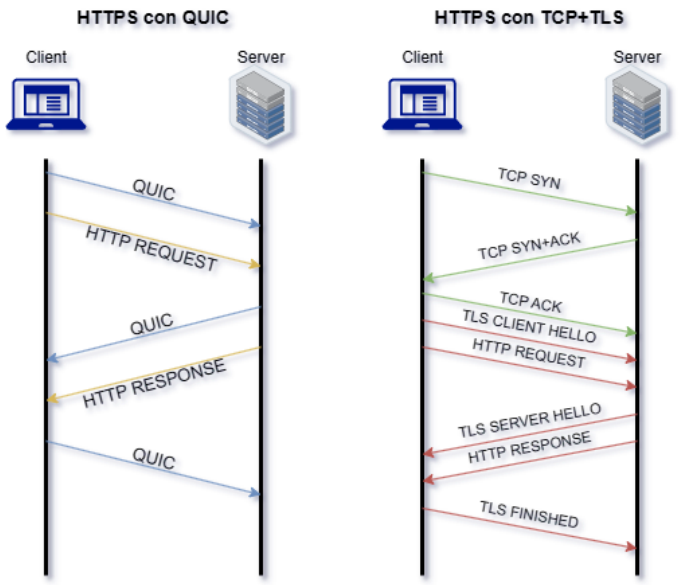


Figura 3.12: Comparazione tra l'handshake con QUIC e con TCP+TLS 1.2

confronta il funzionamento di HTTPS su QUIC e HTTPS su TCP+TLS

HTTPS con QUIC

- **QUIC+HTTP REQUEST/RESPONSE**: Il client utilizza UDP per stabilire una connessione con un solo round-trip time (nei prossimi accessi sarà 0-RTT), nel mentre può inviare richieste HTTP al server e ricevere risposte . Questo viene fatto in parallelo con l'handshake QUIC, riducendo la latenza rispetto all'utilizzo di TCP.

HTTPS con TCP+TLS

Invece, il tradizionale HTTPS su TCP+TLS necessita più passaggi per stabilire una connessione sicura:

- **Handshaking TCP**:
  - **TCP SYN/SYN+ACK/ACK:** il client e il server si scambiano 3 pacchetti per effettuare l'handshake, questo processo richiede 1.5 RTT.
- **Handshaking TLS**: composta da varie fasi tra cui negoziazione della versione di TLS usata, autenticazione del server mediante certificato pubblico, e stabilire la chiave di cifratura della comunicazione.

Sul sito [49] si può visualizzare in modo interattivo e più approfondito il funzionamento dei singoli meccanismi che portano la connessione di un client a un server utilizzando QUIC.
Essendo progettato per essere resiliente ai cambiamenti nelle condizioni di rete, utilizzando identificatori di connessione invece degli indirizzi IP per gestire le connessioni, QUIC può mantenere le connessioni

attive anche se l'indirizzo IP cambia, come nel caso di un dispositivo mobile che passa da una rete Wi-Fi a una rete cellulare. Inoltre, grazie ai meccanismi avanzati di correzione degli errori e al controllo della congestione, QUIC può quindi utilizzare la banda disponibile in modo più efficiente, riducendo il buffering e migliorando la QoE utente. Sono stati fatti anche studi sull'adattamento del bitrate e le architetture di rete avanzate, evidenziando come QUIC possa integrarsi con tecniche di ABR per ottimizzare la QoE in condizioni di rete variabili [50][51]. Come altri protocolli emergenti, QUIC non è ancora universalmente supportato, limitando la sua adozione su larga scala.

3.4 Riduzione della Latenza

La riduzione della latenza è una delle sfide più critiche nel live streaming (LS). Diverse tecniche moderne sono state sviluppate per affrontare questo problema, tra le quali ne troviamo alcune utilizzate dai protocolli appena trattati, come il CTE, che abbiamo già trattato e l'ABR di cui tra poco parleremo più nello specifico.
Altre tecniche, come ad esempio la frame extrapolation(estrapolazione di frame), cercano di compensare una latenza che in alcuni casi è difficilmente riducibile per via di limiti fisici e tecnologici. Nel documento [52] vengono proposti due schemi di compensazione della latenza, uno che prevede l'estrapolazione al lato del codificatore e l'altro al lato del decodificatore. I dati preliminari ricavati, analizzando il compromesso tra latenza e fedeltà utilizzando tre metodi di valutazione, mostrano che è possibile compensare una latenza tipica di 100 ms con una perdita di qualità di 8 dB in *Peak Signal-to-Noise Ratio* (PSNR)(rapporto tra la massima potenza possibile di un segnale e il suo rumore) e l'interpolazione, che sia durante la codifica o la decodifica, produce sempre la stessa perdita di qualità. Il documento si conclude ribadendo l'utilità di un processo di frame interpolation per compensare la latenza in settori come realtà virtuale e cloud gaming e di come ci sia bisogno di ulteriore lavoro negli attuali metodi di interpolazione.

3.4.1 Adaptive Bitrate Streaming (ABR)

Il progressive download, o download progressivo, è una tecnica che permette la riproduzione di un video nel dispositivo dell'utente senza che debba aspettare che l'intero video sia stato scaricato. In linea teorica una volta selezionato il video da visualizzare la riproduzione avviene immediatamente senza interruzioni durante la visione. Per migliorare la QoE del download progressivo sono state adottate tecniche di buffering(pre-caricamento di più segmenti di video) per evitare stalli, però queste tecniche non si adattano ai possibili cambi dinamici delle condizioni di rete, portando spesso a lunghi tempi di caricamento. ABR supera questo ostacolo permettendo al chi trasmette i contenuti di produrre multipli stream di dati a diverse qualità, spaziando da quelle più basse, per dispositivi mobili o con poca larghezza di banda, a quelle più alte per riproduzioni in alta definizione; sincronizzando inoltre la disponibilità di questi diversi stream in modo che se il client decide di passare da un bitrate ad un altro(che sia per scelta personale o per via di un cambio di condizioni di rete) semplicemente inserisce nella coda di riproduzione i nuovi pacchetti a diverso bitrate ottenendo così una transizione fluida e senza interruzioni[38][34].

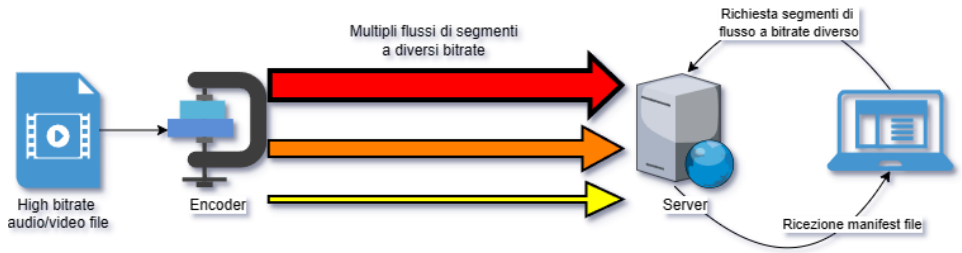


Figura 3.13: Panoramica dell'ABR

Come rappresentato in 3.13 il funzionamento dell'ABR è quanto segue:

- Il processo inizia con un file audio o video, preferibilmente ad alto bitrate.
- **Encoder**: Questo file ad alto bitrate viene inviato a un encoder, che ricordiamo essere un dispositivo o software che può comprimere il file in diversi flussi a bitrate multipli. Ogni flusso rappresenta una versione del video con una diversa qualità.
- **Server**: Il server memorizza questi flussi e gestisce le richieste degli utenti.
- **Richiesta segmenti di flusso a bitrate diverso**: Il dispositivo dell'utente (computer, smartphone, ecc.) richiede segmenti del video al server. Il bitrate di questi segmenti dipende dalla larghezza di banda disponibile e dalle condizioni della rete dell'utente.
- **Ricezione manifest file**: Prima di iniziare a riprodurre il video, il dispositivo dell'utente riceve un file di manifest. Questo file contiene informazioni sui vari segmenti di bitrate disponibili e sulle loro posizioni sul server.
- **Adattamento dinamico**: Durante la riproduzione, il dispositivo dell'utente continua a monitorare la qualità della connessione e può cambiare dinamicamente il bitrate dei segmenti richiesti. Se la connessione è buona, richiede segmenti con un bitrate più alto (migliorando la qualità video); se la connessione peggiora, richiede segmenti con bitrate più basso per evitare interruzioni.

Gli obiettivi generali dell'ABR, dettati dalle necessità di QoE degli utenti, sono l'evitare interruzioni della riproduzione causate dal buffering e massimizzare la qualità video. Tuttavia bisogna trovare un compromesso tra le due in modo di offrire la qualità video più alta in base alle condizioni di rete evitando buffering anziché utilizzare sempre e solo la qualità più bassa. Un altro obiettivo è il ridurre il numero di transizioni da una qualità video all'altra, anche questo però deve trovare un compromesso con il massimizzare la qualità sennò si incappa nel problema di prima, dove utilizzare la qualità più bassa sembra la soluzione. Bisogna quindi non guardare le sfide singole, ma avere una visione di insieme, riuscendo a bilanciare i 4 punti riuscendo così ad offrire una buona QoE per gli utenti.
Artmitage *et al.*[38] ha presentato un documento in cui si discute delle tecniche di ABR su HTTP, analizzando vari meccanismi di adattamento del bitrate; in particolare divide gli algoritmi di ABR dal lato client in 3 macro categorie: *throughput-based, buffer-based e hybrid/control theory based.*

Throughput-Based: questa categoria analizza il "throughput"(ammontare di dati trasferiti in un periodo di tempo) della connessione TCP come segnale di feedback per selezionare le porzioni di video successive. Questi algoritmi si basano principalmente sulla misurazione della velocità di download dei

segmenti video precedenti per stimare la larghezza di banda disponibile e quindi decidere la qualità dei segmenti futuri. Gli algoritmi presenti in questa categoria differiscono soprattutto per il modo in cui misurano il throughput e per come utilizzano i dati misurati. Un esempio di implementazione la troviamo nella ThroughputRule in dash.js(una implementazione open source di riferimento di MPEG-DASH in JavaScript), riassumiamo i passi principali del funzionamento[53]:

Calcolo del throughput medio

```
const throughput = throughputController.getSafeAverageThroughput(mediaType);
const latency = throughputController.getAverageLatency(mediaType);
```

Calcola il throughput medio sicuro e la latenza media per il tipo di media corrente.

Verifica dello stato del buffer e delle metriche

```
if (isNaN(throughput) || !currentBufferState) {
    return switchRequest;
}
```

Controlla se le metriche del throughput e lo stato del buffer sono valide prima di procedere.

Selezione del bitrate ottimale

```
if (abrController.getAbandonmentStateFor(streamId, mediaType) ===
↪  MetricsConstants.ALLOW_LOAD) {
    if (currentBufferState.state === MetricsConstants.BUFFER_LOADED || isDynamic) {
    switchRequest.representation =
    ↪  abrController.getOptimalRepresentationForBitrate(mediaInfo, throughput, true);
        switchRequest.reason = {throughput, latency, message: `[ThroughputRule]:
        ↪  Switching to Representation with bitrate ${switchRequest.representation ?
        ↪  switchRequest.representation.bitrateInKbit : 'n/a'} kbit/s. Throughput:
        ↪  ${throughput}`
        };
    }
}
```

Se lo stato di abbandono del caricamento è permesso e il buffer è caricato (o il flusso è dinamico), viene selezionata la rappresentazione ottimale per il bitrate in base al throughput calcolato. Il calcolo del throughput medio viene effettuato in un altro file[54]:

```
    function getAverageThroughput(mediaType, calculationMode = null, sampleSize =
    ↪  NaN){
        const value = _getAverage(Constants.THROUGHPUT_TYPES.BANDWIDTH, mediaType,
        ↪  calculationMode, sampleSize);
        return Math.round(value); }
```

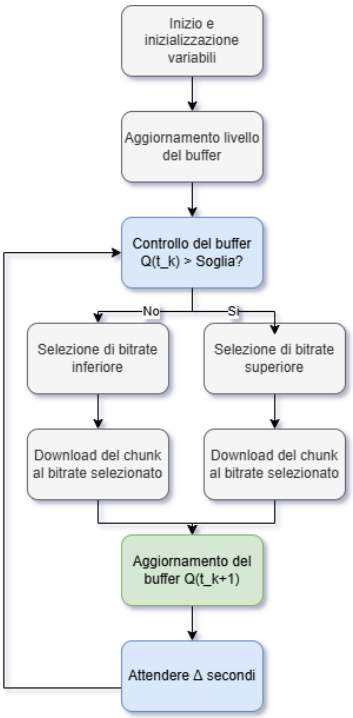
```
function getSafeAverageThroughput(mediaType, calculationMode = null, sampleSize =
↪ NaN) {
    let average = getAverageThroughput(mediaType, calculationMode, sampleSize);
    if (!isNaN(average)) {
        average *= settings.get().streaming.abr.throughput.bandwidthSafetyFactor;
    }
    return average;
}
```

La funzione *_getAverage* è responsabile del calcolo della media del throughput o della latenza per un determinato tipo di media utilizzando vari metodi di calcolo ricevuti in input.

Buffer-Based: gli algoritmi buffer-based utilizzano il livello di riempimento del buffer del lettore video come segnale di feedback per selezionare le porzioni video successive, mirando a mantenere il buffer a un livello ottimale per prevenire interruzioni nella riproduzione video (buffering) .Gli algoritmi buffer-based differiscono soprattutto nel modo in cui monitorano e gestiscono il livello di riempimento del buffer per adattare la qualità del video. Un algoritmo che fa parte di questa categoria è BOLA(Buffer Occupancy based Lyapunov Algorithm), esso utilizza un problema di massimizzazione dell'utilità che incorpora entrambi i principali metriche della Qualità dell'Esperienza (QoE): il bitrate medio del video e la durata del rebuffering. Un aumento del bitrate medio incrementa l'utilità, mentre il rebuffering la riduce. [55]

Il funzionamento può essere sommariamente riassunto in questo schema 3.14:

Figura 3.14: Schema BOLA



- **Inizio e inizializzazione delle variabili**:
  - L'algoritmo inizia con l'inizializzazione delle variabili necessarie per il suo funzionamento. Queste variabili possono includere parametri di configurazione, soglie di buffer e altri valori iniziali richiesti per il calcolo del bitrate ottimale.
- **Aggiornamento del livello del buffer**:
  - Il livello del buffer viene aggiornato per riflettere la quantità corrente di dati video memorizzati. Viene quindi determinato lo stato attuale del buffer, utilizzato per prendere decisioni informate sui successivi chunk di video da scaricare.
- **Controllo del buffer**:
  - L'algoritmo verifica se il livello del buffer $Q(t_k)$ è superiore a una soglia predefinita. Questo controllo è fondamentale per decidere se è necessario scaricare un nuovo chunk video o se il buffer ha già una quantità sufficiente di dati.

- **Selezione del bitrate**:
  - **Selezione di bitrate inferiore** (se $Q(t_k)$ è inferiore alla soglia):
    - Se il livello del buffer è inferiore alla soglia, l'algoritmo seleziona un bitrate inferiore per evitare il rebuffering.
  - **Selezione di bitrate Superiore** (se $Q(t_k)$ è superiore alla soglia):
    - Se il livello del buffer è superiore alla soglia, l'algoritmo può selezionare un bitrate superiore, dato che c'è un margine di sicurezza sufficiente per gestire eventuali variazioni nella larghezza di banda senza causare rebuffering.
- **Download del chunk al bitrate selezionato**:
  - Una volta selezionato il bitrate ottimale, l'algoritmo scarica il chunk video al bitrate scelto.
- **Aggiornamento del Buffer** $Q(t_{k+1})$:
  - Dopo il download del chunk ottimale, il livello del buffer viene aggiornato nuovamente per riflettere l'aggiunta del nuovo dato. Questo aggiornamento è essenziale per il controllo continuo del buffer e per le decisioni future sul bitrate.
- **Attendere Δ Secondi**:
  - L'algoritmo attende per un intervallo di tempo Δ prima di ripetere il controllo del buffer.

Hybrid/control-theory: in quest'ultima categoria vengono inseriti gli algoritmi che, come indicatori delle condizioni di rete, fanno utilizzo del controllo basato sia sul throughput, sia sul buffer. PANDA (Probe AND Adapt) è un algoritmo che utilizza un approccio di controllo proporzionale-integrale-derivativo (PID) per bilanciare la qualità del video e la stabilità del buffer.

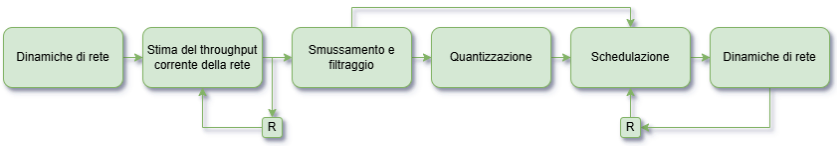


Figura 3.15: Schema algoritmo PANDA

Facciamo ora una analisi dello schema di funzionamento dell'algoritmo in figura 3.15

1. **Stima della Condivisione della Larghezza di Banda**: L'algoritmo stima la larghezza di banda corrente disponibile ($\hat{x}[n]$) utilizzando i dati di throughput dei segmenti di download precedenti.
2. **Smussamento della Stima**: La stima del throughput viene filtrata per ridurre il rumore, ottenendo una versione smussata $\tilde{y}[n]$. Questo processo di smussamento aiuta a eliminare i valori anomali e a stabilizzare la stima.
3. **Quantizzazione**: La stima smussata del throughput $\tilde{y}[n]$ viene mappata a un bitrate video discreto $r[n]$ tra quelli disponibili. Questa quantizzazione converte il valore continuo della stima in uno dei bitrate utilizzati per lo streaming video.

4. **Schedulazione della Prossima Richiesta di Download**: L'algoritmo determina l'intervallo di tempo target $\hat{T}[n]$ fino alla prossima richiesta di download, basato sul bitrate discreto scelto $r[n]$, la stima throughput smussata $\tilde{y}[n]$e sullo stato del buffer $B[n]$, con l'obiettivo di portarlo il più vicino possibile ad una soglia minima determinata. La schedulazione considera il tempo necessario per il download e il livello di riempimento del buffer, per mantenere un equilibrio tra qualità e stabilità.

I loop presenti negli step di stima e scheduling servono, rispettivamente, per poter determinare il bitrate medio $\hat{x}$ e il tempo di iter-richiesta $\hat{T}$, una peculiarità di questo algoritmo è l'architettura ibrida poiché si può sia utilizzare con i loop chiusi, che con i loop aperti, per ridurre i cambi di bitrate dovuti alla quantizzazione rendendo così possibile la stabilizzazione dei valori stimati di $\hat{x}$[56].

Capitolo 4

Codec e Trasmissione per il Live Streaming

4.1 Introduzione alla Codifica e Compressione Video

Un secondo di video in alta definizione (1920*1080p) con un framerate di 30 fotogrammi occupa circa 250MB di spazio di archiviazione, se questo file durasse un minuto raggiungerebbe una dimensione di 450GB, è evidente come lo spazio occupato sia enorme e di come la larghezza di banda necessaria per trasmettere 250MB/s di video non compresso, non includendo audio e informazioni aggiuntive sia difficilmente raggiungibile per un utente qualunque[57]. Serve quindi un codificatore, ossia un dispositivo fisico o un programma che comprime i dati per ridurre la loro dimensione e li decomprime(in tal caso si parla di decoder) quando è necessario. Nel contesto dei video, i codec sono essenziali per gestire le enormi quantità di dati generati dai video non compressi. La codifica video trasforma i segnali video non compressi in un formato compresso per la trasmissione su reti a larghezza di banda limitata. I codec più comuni, come abbiamo già visto, includono H.264, H.265 (HEVC) e VP9. H.264 è ampiamente utilizzato per la sua efficienza e qualità accettabile, mentre H.265 offre una compressione superiore ma richiede più potenza di elaborazione. VP9, sviluppato da Google, è un'alternativa open source che offre una buona qualità di compressione. La maggior parte dei codec sono lossy codec, ossia parte delle informazioni audio e/o video vengono perse durante la codifica, generalmente più compressione si ottiene corrisponde più qualità si perde. Questo processo di codifica comprime il video rimuovendo anche informazioni video che si ripetono. Ad esempio, se lo sfondo di una scena rimane statico mentre il soggetto principale è fermo o quasi immobile, non è necessario renderizzare e comprimere lo sfondo in ogni frame. Si possono applicare tecniche di compressione temporale e spaziale per ottimizzare l'efficienza. La compressione temporale sfrutta la ridondanza temporale, che si riferisce alle somiglianze tra frame successivi in una sequenza video. Poiché molti elementi in una scena video possono rimanere invariati da un frame all'altro, è inefficiente codificare ogni frame completamente da zero. La compressione temporale codifica lo sfondo una volta e successivamente registra solo le differenze rispetto ai frame precedenti. Questo approccio sfrutta la predizione temporale del movimento, riducendo la quantità di dati necessari per la trasmissio-

ne e aggiornando lo sfondo solo quando viene rilevato un cambiamento significativo. Parallelamente, la compressione spaziale sfrutta la ridondanza spaziale, ossia le somiglianze all'interno di un singolo frame. All'interno di un frame, infatti, ci possono essere ampie aree con colori simili. La compressione spaziale utilizza tecniche come la trasformata discreta del coseno (*Discrete Cosine Transform*) per identificare e ridurre le ridondanze nelle regioni omogenee dell'immagine, codificando efficientemente le aree uniformi o con variazioni minime. Combinando queste due tecniche, si riduce la larghezza di banda necessaria senza compromettere eccessivamente la qualità visiva[58].
La codifica video live può essere descritto come un sottotipo della codifica video impiegata per la trasmissione in tempo reale. Vista la necessità di codificare e trasmettere dati con latenza minima, risulta più complessa e con necessità diverse rispetto alla semplice codifica video. Come ben sappiamo le sfide più importanti sono latenza, qualità video, scalabilità e compatibilità.

4.1.1 Tecniche di Trasmissione e Streaming

Ogni piattaforma di LS gestisce gli streaming a propria discrezione. Twitch, una delle piattaforme di LS più popolari al mondo, affronta diverse sfide tecniche per garantire un'esperienza di streaming fluida e di alta qualità per milioni di spettatori[5]. Per ridurre il ritardo tra la trasmissione e la visualizzazione e consentire interazioni in tempo reale, Twitch, in aggiunta all'uso del tradizionale HLS [59], ha implementato una modalità a bassa latenza sfruttando il LL-HLS [60] che riduce il ritardo a pochi secondi, a patto che si abbia una buona connessione. Per mantenere una qualità video elevata nonostante le variazioni della rete Twitch riceve dallo streamer il file codificato e con bitrate fisso, una volta nei propri server lo stream viene transcodificato, utilizzando RTMP, in multiple stream HLS (e LL-HLS)[61] a bitrate diversi così da poter adattare dinamicamente la qualità del video in base alle condizioni di rete in tempo reale, evitando buffering e interruzioni.
Per supportare milioni di spettatori simultanei senza degradare il servizio, Twitch utilizza una vasta rete di CDN, distribuendo così il carico di traffico e migliorando la velocità di consegna dei dati si riducono i tempi di caricamento e minimizzano i buffering. Inoltre, l'uso di server di EC per elaborare e trasmettere i dati vicino agli utenti finali riduce la latenza e migliora l'efficienza complessiva della rete. Infine, per assicurare che lo streaming funzioni su una vasta gamma di dispositivi, Twitch transcodifica i flussi in vari formati compatibili con desktop, mobile e console, garantendo che gli spettatori possano accedere ai contenuti una vasta gamma di dispositivi senza problemi.

4.2 Decodifica e Innovazioni nei Codec Video

Come già discusso nel primo capitolo la decodifica consiste nel demultiplexing dello stream di dati nei singoli flussi audio e video codificati, nella decompressione e infine sincronizzazione dei flussi audio/video attraverso i timestamp.
Come nel caso della codifica, soprattutto se in applicazioni di LS, il processo deve essere efficiente in modo da ridurre la latenza finale non introducendo ritardi aggiuntivi troppo elevati. L'aumentare dei costi di archiviazione e del trasporto di dati ha portato alla necessità di sviluppare standard di codifica

sempre più ottimizzati e con meno necessità di bitrate a parità di qualità.

4.2.1 Standard di Codifica e Compressione: Da H.264 a AV1

Per anni lo standard di codifica è stato l'H.264/AVC[62]. Nel 2013 sono stati introdotti nuovi standard, come VP9[63] e HEVC(High Efficency Video Coding)[64], i quali offrono risparmi di bitrate medi del 50%[65] rispetto a quelli necessari per H.264/AVC. Tuttavia, H.264/AVC continua a essere ampiamente utilizzato. Nonostante le avanzate capacità di compressione, VP9 e HEVC non hanno soppiantato H.264/AVC e la loro adozione è ancora poco diffusa.
Con l'obiettivo di superare le problematiche legate ai costi di licenza e alla sempre maggiore necessità di standard improntati all'efficienza della compressione video nel 2015 è stato fondato il consorzio *Alliance for Open Media*(AOMedia), composto da un gruppo di aziende tecnologiche di primo piano, tra cui Amazon, Cisco, Google, Intel, Microsoft, Mozilla e Netflix. Han *et al.* [66] fornisce un'analisi tecnica del formato di compressione AV1, spiegandone il design e confrontandolo con il suo predecessore VP9, infine ne valuta le prestazioni. Le differenze con i suoi predecessori H.264 e VP9 sono numerose, cerchiamo di presentarne quattro tra le principali[67]:

Supporto per Varie Rappresentazioni di Pixel

- **Formati e profondità**: mentre VP9 e H.264 supportano il formato di colore 4:2:0, AV1 supporta più formati di colore inclusi 4:0:0 (monocromatico), 4:2:0, 4:2:2 e 4:4:4. Questo permette una maggiore flessibilità e adattabilità in diverse applicazioni, supporta inoltre rappresentazioni di pixel fino a 12 bit, contro i massimi 10 di VP9. La capacità di supportare 10 e 12 bit è necessaria per la produzione di contenuti HDR (High Dynamic Range), offrendo una gamma dinamica più ampia e una maggiore precisione del colore.

Tecniche di Predizione e Codifica

- **Predizione Intra-frame**: AV1. rispetto i predecessori, include più modalità di predizione direzionale con maggiore granularità e intrablock copy (previsione del movimento all'interno dello stesso frame), migliorando la capacità di predire con precisione i contenuti statici e dinamici.
- **Predizione Inter-frame**: Supporta la compensazione del *translational motion* e dell'*affine motion*. La compensazione dell'affine motion consente al codec di gestire meglio movimenti complessi come rotazioni e cambi di scala, offrendo una qualità predittiva superiore rispetto agli altri che utilizzano principalmente solo il translational motion.
- **Predizione Composta**: offre varie modalità di predizione composta, permettendo la combinazione di previsioni da più frame di riferimento per migliorare la qualità e la stabilità della previsione. Questo include tecniche come la compensazione del movimento bidirezionale e la previsione con maschera a cuneo(utilizzo di una maschera binaria per unire due previsioni diverse su uno stesso blocco di pixel dell'immagine, contribuendo all'efficienza della compressione)

Blocco di Codifica e Partizionamento

AV1:

- **Superblocchi**: AV1 introduce il concetto di superblocco con dimensioni fino a 128×128 pixel(divisi in una serie di blocchi da 64x64), il doppio rispetto a VP9 e otto volte rispetto a H.264, permettendo una maggiore flessibilità nel partizionamento dei blocchi e migliorando l'efficienza della compressione adattando la codifica alle caratteristiche specifiche del contenuto video.
- **Partizionamento**: Permette una partizione ricorsiva dei blocchi, che consente di adattare la codifica a diverse caratteristiche del contenuto video, migliorando la precisione della compressione per aree diverse del video.

Considerazioni Hardware

AV1:

- **Ottimizzazione per Hardware**: Progettato con particolare attenzione alla fattibilità hardware, AV1 cerca di bilanciare le prestazioni di compressione con la complessità di decodifica per garantire che possa essere efficacemente implementato su dispositivi mobili e hardware dedicato. Ad esempio, includendo tecniche che considerano la latenza e l'area del silicio necessaria per la decodifica.
- **Decodifica Hardware**: Molti strumenti di codifica di AV1 sono progettati per essere efficienti dal punto di vista hardware, garantendo che le operazioni possano essere eseguite senza eccessivo consumo di risorse.

AV1 si presenta come il prossimo codec di riferimento per le piattaforme di LS, grazie alla grande riduzione del bitrate rispetto ai predecessori senza compromettere la qualità video dopo la compressione. All'inizio del 2024 Twitch ha, infatti, annunciato l'inizio del beta testing dell'utilizzo AV1 come codec per il LS[68].

Capitolo 5

Qualità dell'Esperienza (QoE)

Nei capitoli precedenti è stata numerose volte utilizzata la terminologia *Quality of Experience* (QoE) limitandosi a definirla genericamente come misura del grado di soddisfazione durante l'utilizzo di un servizio, in questo capitolo verrà data una definizione più precisa del termine, del suo significato e delle tecniche e metodologie per determinarla.

5.1 Definizione e metodologie di misurazione

La *International Telecommunication Union* (ITU) da la seguente definizione standardizzata di QoE[69], formalizzata inizialmente dal Qualinet White Paper[70]:

The degree of delight or annoyance of the user of an application or service

Possiamo quindi descrivere la QoE come Il grado di appagamento provato dall'utente finale di un'applicazione o di un servizio durante il suo utilizzo. Sensazione derivante dal soddisfacimento delle sue aspettative in confronto all'utilità, dalla fruizione dell'esperienza alla luce della personalità dell'utente e del contesto in cui si trova. Un'altra terminologia spesso utilizzata insieme alla QoE è la *Quality of Service* (QoS); nell'ambito delle telecomunicazioni viene utilizzata per identificare la qualità e l'efficacia di un servizio da un punto di vista oggettivo, senza però tenere conto dei fattori legati alla percezione effettiva dell'utente, caratteristica della QoE. Il termine QoE, ha quindi un significato olistico, incorporando la qualità del servizio come sottoinsieme ed espandendola tenendo conto anche dei fattori umani, come lo stato d'animo e il contesto in cui si fruisce del contenuto.

5.1.1 Valutazione oggettiva e soggettiva

La valutazione della qualità video (Video Quality Assesment) può essere definita in due categorie: oggettiva e soggettiva. Oggettiva viene utilizzata una misurazione, mediante modelli matematici standardizzati e regolati, di metriche che cercano di rappresentare la qualità dell'immagine o del video. Soggettiva quando si cerca di misurare la qualità video mediante il riscontro degli utenti. Anche le valutazioni del contenuto da parte degli utenti, come nel caso delle valutazioni oggettive, sono soggette al rispetto di linee guida standardizzate[71][72]. In questo modo si riesce ad effettuare una misurazione ripetibile e

Figura 5.1: Comparazione della stessa immagine con diversi tipi di distorsione con lo stesso MSE

affidabile, quantizzata in un *Mean Opinion Score* (MOS), ossia una media delle varie valutazioni ottenute durante il processo.

Entrambe le valutazioni hanno i loro svantaggi. Infatti per quanto le valutazioni soggettive forniscano misurazioni della effettiva qualità dell'esperienza utente, non rappresentano effettivamente l'esperienza generale. Questo accade perché, per effettuare questi test, vengono utilizzati campioni di utenti che, per quanto grandi ed eterogenei nella composizione, non riuscirà mai a rappresentare l'effettiva totalità della utenza finale. Inoltre questi test sono dispendiosi sia in termini di tempistiche che di risorse finanziarie, riuscendo a trattare e valutare solo alcuni fattori di influenza, mentre altri, per cui servirebbero molto tempo e risorse, non vengono valutati.

Le valutazioni oggettive utilizzano metriche come il PSNR e il *Structural Similarity Index Measure* (SSIM), queste metriche però non sempre riescono nel determinare quale sia il video o l'immagine di miglior qualità e a rappresentare la percezione di qualità dell'utente finale. Ad esempio una stessa immagine può essere soggetta a diverse distorsioni ma presentare sempre lo stesso *Mean Square Error* (MSE)(discrepanza quadratica media fra i valori dei dati osservati ed i valori dei dati stimati, da questo valore si ricava il PSNR), ma qualità percettive molto diverse[73]. Nell'immagine 5.1 l'immagine originale viene alterata e distorta in vari modi, conservando però sempre lo stesso MSE(quindi anche il PSNR)[73]. Lo stesso problema affligge anche il SSIM. Per poter calcolare queste metriche di valutazione è necessario avere a disposizione, oltre che all'immagine o al video modificato, anche quello originale e con cui effettuare la comparazione, non sempre disponibile in scenari di vita reale. Queste metriche sono state sviluppate con in mente la misurazione di perdita di qualità dovuta alla perdita di pacchetti durante la trasmissione o alla compressione del contenuto multimediale,. Non tengono quindi conto di altri fattori di influenza della qualità, come ad esempio il buffering o il cambio di qualità durante la riproduzione.

Sono quindi necessari nuovi modelli e tecniche di misurazione della QoE che tengano conto dei nuovi indici di valutazione dell'esperienza introdotti con l'ascesa dello streaming adattivo come nuova frontiera del lavoro e dell'intrattenimento[74].

5.1.2 Fattori di influenza

Risulta molto complesso determinare dei fattori di influenza, o *Influence Factor* (IF)s, globali e validi per ogni tipo di contenuto o servizio multimediale. Come riportato nella tabella 5.1 vari studi hanno proposto diverse classificazioni dei IFs in base al tipo di contenuto.

Authors	**Models**
M. Yang *et al.*[75]	Objective factors: system factors, context factors; Subjective factors: human factor
Juluri *et al.*[26]	System level, Context level, User level, Content level
Baraković *et al.*[76]	Technology performance, usability, subjective evaluation, expectations, context
Skorin-Kapov e Varela[77]	Application, Resource, Context, User spaces
Stankiewicz e Jajszczyk[78]	Quality of Service (QoS) factors, Grade of Service (GoS) factors, Quality of Resilience (QoR) factors, non-technology related factors

Tabella 5.1: Modelli di fattori di influenza della QoE

Laiche *et al.*[79], basandosi sulla definizione di QoE[69] e sugli studi presentati nella tabella 5.1, presenta un modello di classificazione degli IFs nei servizi video suddiviso in quattro categorie: *system factors, context factors, human factors, e service factors.*

1. Fattori di Sistema (System Factors)

I fattori di sistema includono parametri tecnici legati al dispositivo utilizzato e alla rete. Questi parametri determinano la qualità prodotta di un servizio o applicazione. Esempi di tali fattori includono:

- **Delay (Ritardo)**: Il tempo di latenza nella trasmissione dei dati.
- **Jitter**: La variazione del ritardo dei pacchetti.
- **Bandwidth (Larghezza di banda)**: La capacità della rete di trasmettere dati.
- **Throughput (Portata)**: La velocità effettiva di trasmissione dei dati.
- **Loss rate (Tasso di perdita)**: La percentuale di pacchetti persi durante la trasmissione.

2. Fattori Contestuali (Context Factors)

I fattori contestuali rappresentano le dimensioni che indicano la posizione fisica e lo spazio, nonché le relazioni interpersonali esistenti durante l'esperienza di visione. Questi fattori possono includere:

- **Physical context IFs**: Caratteristiche dell'ambiente come il luogo e le condizioni ambientali.
- **Temporal context IFs**: Aspetti temporali come l'ora del giorno, la durata del video e il comportamento mantenuto durante la fruizione del contenuto.
- **Social context IFs**: Esperienza sociale determinata dalle relazioni interpersonali e dalla partecipazione di altre persone durante la visione.
- **Economic context IFs**: Aspetti economici come il tipo di abbonamento e il costo finanziario.
- **Task context IFs**: Tipo di esperienza, può essere singola, multipla o essere una esperienza di interruzione(ad esempio una notifica o un pop-up).

- **Technical and information context IFs**: Relazione tra il sistema di interesse e altri sistemi, descrive come il sistema principale (es. servizio di streaming video) interagisce con altri sistemi e componenti tecnici.

3. Fattori Umani (Human Factors)

I fattori umani possono essere suddivisi in due categorie principali: fattori di elaborazione di basso livello e fattori di elaborazione di alto livello.

- **Low-level processing IFs**: Caratteristiche dell'utente come genere, età e stato emotivo.
- **High-level processing IFs**: Aspetti socio-culturali, posizione socio-economica, background educativo e fase della vita dell'utente.

4. Fattori di Servizio (Service Factors)

I fattori di servizio possono essere suddivisi in due parti: fattori oggettivi e fattori soggettivi.

- **Objective factors**: Includono i fattori relativi ai contenuti e ai media.
  - **Content factors**: Affidabilità dei contenuti e tipo di contenuti come risoluzione spaziale e temporale(rispettivamente la capacità di distinguere singoli elementi tra loro ravvicinati spazialmente e il tempo minimo necessario per registrare un fenomeno che si sta osservando) e velocità del movimento.
  - **Media factors**: Configurazione dei media come la codifica e il frame rate, che possono variare in base allo stato della rete durante la trasmissione.
- **Subjective factors**: Complessi e di natura non misurabile rispetto ai fattori oggettivi, sono importanti per la valutazione della QoE poiché riflettono la percezione dell'utente.
  - **User engagement**: Caratteristiche dell'investimento dell'utente nei servizi video, come tempo e attenzione.
  - **User behavior**: Azioni dell'utente in risposta al consumo del servizio di streaming video, come mettere in pausa un video o cliccare su regioni non interattive del player.
  - **Social-related status**: Interazioni sociali in relazione al servizio video e all'esperienza di pre-visione, come valutazioni dell'utente, commenti e numero di visualizzazioni.

5.2 Tecniche di miglioramento della QoE

Nei capitoli precedenti, abbiamo presentato e sottolineato l'importanza dell'EC e del MEC nel migliorare la QoE spostando l'elaborazione dei dati più vicino agli utenti finali. Abbiamo anche parlato di come l'utilizzo di CDN possa ridurre la latenza e migliorare la velocità di caricamento dei contenuti. Tuttavia, esistono ulteriori tecniche e tecnologie che possono essere impiegate per ottimizzare ulteriormente la QoE in situazioni di streaming live o real-time. In questo sottocapitolo, vedremo alcune altre tecniche che aiutano a migliorare la QoE attraverso nuovi approcci.

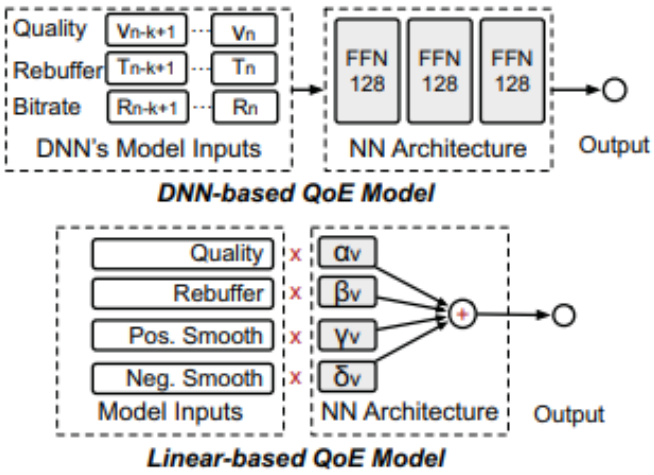


Figura 5.2: Architettura della rete neurale[80]

5.2.1 Ottimizzazione basata sul feedback

Molti algoritmi di ABR utilizzano modelli di QoE che sfruttano il MOS per ottimizzare la qualità percepita del video. Questi modelli, come ben sappiamo, non riescono a rappresentare ed integrare in modo efficiente la diversità delle scale di valutazione degli utenti in ambienti reali. Nel documento "Optimizing Adaptive Video Streaming with Human Feedback" Huang *et al.*[80] presentano un approccio innovativo chiamato *Jade* che, sfruttando l'apprendimento per rinforzo da feedback umano, *Reinforced Learning with Human Feedback* (RLHF), ottimizza gli algoritmi di ABR.

Modello di QoE Basato su Ranking

Con l'obiettivo di superare i limiti naturali dei modelli di QoE basati sul MOS, Jade implementa un modello basato su un ranking ottenuto normalizzando i punteggi derivanti dalle opinioni degli utenti. Questo processo di allineamento dei punteggi permette di aggregare e confrontare le valutazioni su una scala comune. Per fare ciò il modello utilizza due architetture neurali, una lineare e una profonda, come mostrato in figura 5.2:

- **Rete Neurale Lineare:** Utilizzata nelle fasi iniziali dell'addestramento, grazie alla sua semplicità, essendo composta da 4 neuroni, e alla bassa necessità computazionale rispetto alla rete neurale profonda, permette di ottenere una rapida convergenza dei risultati ottenendo una valutazione preliminare del modello di QoE.
- ***Deep Neural Network* (DNN):** Questa rete neurale profonda è più complessa e composta da 3 layer *Feed Forward Network* (FFN) da 128 neuroni ciascuno , capace di rilevare relazioni non lineari nei dati di feedback degli utenti, migliorando la precisione delle previsioni di QoE.

La metodologia di Jade prevede:

- **Fase di valutazione dell'utente**: gli utenti forniscono punteggi relativi per le sessioni video, che vengono utilizzati come input per il modello di QoE basato su ranking.
- **Fase di apprendimento del modello di QoE**: Utilizzando il feedback degli utenti, vengono addestrati due tipi di modelli di QoE, uno lineare e uno basato su DNN.

- **Generazione degli algoritmi ABR**: Con l'ausilio del modello di QoE addestrato, Jade utilizza tecniche di apprendimento per rinforzo per generare algoritmi ABR basati su reti neurali.

Meccanismi di Apprendimento

Jade incorpora due meccanismi basati sull'entropia: l'apprendimento fluido(smooth training) e la selezione delle tracce online(online trace selection), per sfruttare appieno i vantaggi dei modelli di QoE. L'apprendimento fluido utilizza il modello lineare di QoE nelle prime fasi dell'addestramento, per poi passare al modello DNN nelle fasi successive, garantendo un'integrazione senza soluzione di continuità tra i due modelli unendo così la rapidità iniziale alla precisione derivante dalla DNN. La selezione delle tracce online adotta tecniche di apprendimento online per scegliere dinamicamente la traccia di rete appropriata per l'addestramento, prima monitorando le condizioni di rete e raccogliendo il feedback degli utenti, poi selezionando la traccia video ottimale, creando così un processo di apprendimento iterativo e continuo.

Secondo i risultati sperimentali, Jade supera di gran lunga gli algoritmi di ABR esistenti in condizioni di rete lente e veloci, migliorando la QoE di almeno il 23.07% rispetto ad altri algoritmi basati sulla QoE. Jade è in grado di bilanciare il compromesso tra la qualità e lo stall ratio(proporzione del tempo di visione di un video speso in buffering). In aggiunta, l'utilizzo e l'unione di modelli di QoE basati sia su reti neurali lineari che DNN permette a Jade di ottenere risultati superiori rispetto agli approcci che utilizzano esclusivamente uno solo dei due modelli.

L'approccio di Jave pone sotto i riflettori il ruolo importante che ha il tenere conto dell'eterogeneità degli utenti nei modelli QoE e mostra la possibilità di combinare il feedback umano con tecniche di apprendimento rinforzato per migliorare la QoE nello streaming video adattivo. Questo metodo rappresenta un significativo passo avanti nella personalizzazione dell'esperienza di streaming in base alle preferenze degli utenti, offrendo una qualità di visione superiore e più soddisfacente.

5.3 Delay Compensation

Il delay compensation, o compensazione del ritardo, è un insieme di tecniche e metodologie per prevenire e mitigare gli effetti che la latenza di rete può avere sulla QoE del cloud e online gaming. La latenza, o ritardo, in questo caso indica il tempo che passa tra l'azione da parte dell'utente di premere un pulsante e la risposta da parte del personaggio sullo schermo. Una latenza alta influenzerà negativamente l'esperienza di gioco perché i ritardi nelle risposte produrranno una diminuzione della reattività, della fluidità percepita e della complessiva precisione e accuratezza dei movimenti.

5.3.1 Delay compensation per il gaming online

Con il termine "Gaming Online" ci si riferisce ai videogiochi in cui il client esegue gran parte del calcolo grafico e della logica di gioco, mentre si comunica con un server centrale per sincronizzare le azioni dei giocatori connessi. Le tecniche di delay compensation nel gioco online sono basate su metodi come *Delayed Input*, *Time Off-setting* e *Prediction*, [74]. Le "Delayed Input Techniques" sono metodi utilizzati per mitigare gli effetti del ritardo di rete nei giochi multiplayer online, cercando di offrire un'esperienza

di gioco più uniforme e sincronizzata tra tutti i giocatori. Introducendo intenzionalmente un ritardo nei comandi dei giocatori locali per allinearsi meglio con i comandi ricevuti dagli altri giocatori attraverso la rete riducendo le discrepanze dovute alla latenza. Esempi di questa tecnica sono "bucket syncronization" e "local lag", nel primo il tempo di gioco è diviso in intervalli (o "bucket") di durata fissa. Ogni comando dei giocatori viene raccolto e ritardato fino alla fine del bucket corrente prima di essere eseguito. Questo significa che tutti i comandi raccolti durante lo stesso intervallo vengono eseguiti simultaneamente alla fine dell'intervallo, garantendo che le azioni dei giocatori siano sincronizzate. Nel secondo, invece, si ritarda intenzionalmente l'esecuzione dei comandi locali per allinearsi meglio con i comandi ricevuti dagli altri giocatori. Per esempio, se un giocatore invia un comando, questo viene ritardato di un certo tempo(solitamente nell'ordine dei ms) prima di essere effettivamente eseguito nel gioco. Questo tempo di ritardo è scelto in modo tale da permettere la ricezione dei comandi dagli altri giocatori, garantendo che tutte le azioni siano eseguite simultaneamente.
Con le "Time-Offsetting Techniques" si cerca di compensare i ritardi di trasmissione temporale, memorizzando un'istantanea dello stato del gioco in momenti diversi e rendendo possibile il ripristino allo stato precedente del gioco in caso di incoerenza tra un evento avvenuto dal lato client che, però, per via della latenza introdotta durante la trasmissione al server, non corrisponde più allo stato attuale del gioco dal lato server. In questo modo gli eventi vengono rilevati in base a quando vengono registrati sul client anziché quando arrivano sul server.
Alle "Prediction Techniques" dedichiamo qualche riga ulteriore, presentando anche un caso di studio specifico.

Prediction Techniques

La compensazione del ritardo dal lato client (Client-Side Delay Compensation) viene utilizzata per mitigare gli effetti della latenza di rete. Il client prevede l'esito delle azioni del giocatore prima di ricevere la conferma dal server. Ad esempio, quando un giocatore muove il proprio avatar, il client visualizza immediatamente il movimento. Nel documento [81], Watanabe *et al.* propone un sistema innovativo di compensazione del ritardo lato client. Nei giochi multiplayer online, il ritardo di rete può portare a gravi errori di sincronizzazione, influenzando negativamente l'esperienza di gioco e le prestazioni dei giocatori. Il problema principale risiede nel fatto che le posizioni dei giocatori e altre informazioni di gioco non sono aggiornate in tempo reale sui dispositivi di tutti i giocatori connessi, causando discrepanze e vantaggi ingiusti. Gli attuali metodi di compensazione del ritardo, come la Lag Compensation(compensazione del lag), operano a livello di server e non riescono a correggere queste discrepanze a livello dei dispositivi dei giocatori.
Il sistema proposto è costituito da due componenti principali: un metodo basato sulla regressione lineare e uno sul *Deep Reinforced Learning* (DLR). L'obiettivo è ridurre gli errori di sincronizzazione predicendo accuratamente la posizione attuale e i comandi dei giocatori.

Il metodo di regressione lineare utilizza le posizioni passate del giocatore per un numero determinato di frame per prevedere la posizione futura. Questo approccio assume che i movimenti dei giocatori seguano un modello prevedibile che può essere catturato tramite una regressione lineare. Questo metodo è computazionalmente efficiente e semplice da implementare, rendendolo adatto per sistemi con risorse

limitate. Tuttavia, la sua accuratezza dipende dalla linearità dei movimenti dei giocatori; movimenti complessi o irregolari possono ridurre significativamente l'efficacia del metodo.

Il metodo basato su DLR utilizza un'architettura di rete neurale convoluzionale per estrarre caratteristiche spaziali e temporali da "snapshot"(rappresentazioni istantanee catturate a intervalli regolari) di campi di gioco passati. Il metodo DRL si sviluppa attraverso i seguenti passaggi:

1. I dati di input consistono in quattro snapshot consecutivi del campo di gioco, convertiti in immagini in scala di grigi con una risoluzione di 84x84 pixel.
2. Le immagini sono elaborate tramite tre strati convoluzionali seguiti da funzioni di attivazione per estrarre le caratteristiche rilevanti.
3. Le caratteristiche estratte sono utilizzate per prevedere i comandi futuri del giocatore. Il modello è addestrato utilizzando funzioni di ricompensa nel caso in cui la predizione sia giusta.

Il metodo DLR è altamente efficace nel gestire movimenti complessi e irregolari grazie alla sua capacità di apprendere dinamiche intricate dai dati. Tuttavia, richiede una notevole potenza computazionale e un lungo tempo di addestramento, rendendolo meno adatto per dispositivi con risorse limitate.

Questa tecnica di compensazione rappresenta un passo avanti nella mitigazione degli effetti del ritardo di rete nei giochi "sparatutto" multiplayer online, grazie all'utilizzo del machine learning. Gli autori riconoscono che, tuttavia, ci sono due principali aree di miglioramento:

- Complessità Computazionale: La predizione successiva basata su DLR attualmente non soddisfa i requisiti di utilizzo in tempo reale per ritardi di rete elevati. Ottimizzare la complessità computazionale rimane un obiettivo futuro.
- Requisiti di Sistema: La soluzione proposta richiede API specifiche per estrarre e aggiornare le informazioni di gioco, il che potrebbe non essere sempre supportato dai giochi online esistenti. L'adattamento del sistema per ambienti di gioco reali sarà un'area chiave per future ricerche.

5.3.2 Delay compensation per il cloud gaming

Come evidenziato dalla letteratura esistente[74] e confermato durante la stesura della tesi, gli studi sulle tecniche di compensazione del ritardo nel cloud gaming, ossia videogiochi che vengono eseguiti su un server remoto in un data center a cui l'utente può giocare collegandosi attraverso il suo dispositivo, sono ancora piuttosto limitati e preliminari.

Outatime[82]

È una tecnica che cerca di prevedere la prossima azione dell'utente basandosi sui comportamenti passati e sulle tendenze recenti delle sue azioni, prevedendo diversi possibili esiti del gioco e renderizzando i frame in anticipo secondo questi esiti. Successivamente, consegna più frame per le azioni anticipate. Inoltre, è previsto un sistema di rollback(che permette di riportare al gioco a uno stato precedente) a costo di scatti durante il gioco, nel caso di previsioni errate. Sebbene questa tecnica possa essere molto efficace nel ridurre l'effetto del ritardo, l'invio di più frame può aumentare notevolmente la larghezza di banda e la capacità computazionale richiesta, creando ulteriori problemi come aumento della perdita di pacchetti,

riduzione della qualità video, ritardo e costi aggiuntivi. Inoltre, simile ai giochi online, l'uso di tecniche predittive può causare scene a scatti in caso di previsioni errate.

Input ritardato[83]

La tecnica prevede l'aggiunta di ritardo ai dispositivi dei client a bassa latenza per bilanciare quella degli utenti ad alta latenza e aumentare l'equità nei giochi multiplayer. Sebbene l'applicazione di questo metodo potrebbe aumentare l'equità complessiva, sicuramente diminuirebbe la qualità dell'esperienza dei giocatori a bassa latenza, obbligati a giocare con un ritardo imposto. Invece, un meccanismo di matchmaking(connessione dei giocatori a server di gioco online) che abbina utenti con lo stesso livello di ritardo potrebbe essere più efficace, mantenendo lo stesso ritardo e garantendo una dinamica di gioco equa.

Time Warp[84]

La tecnica di "time warp" nel cloud gaming è molto simile a quella di "time off-setting" trattata in precedenza: Infatti, allo stesso modo, rileva il successo di un evento in base al tempo registrato sul client dell'utente, non al tempo di arrivo nel server. Ad esempio, in un evento come il colpire un bersaglio sparandogli in un videogioco, quando il server riceve i dati relativi alla posizione del colpo, anziché confrontare la posizione con lo stato attuale del gioco, la confronta con lo stato del gioco al momento esatto in cui il colpo è stato sparato secondo il client dell'utente. In questo modo, anche se il bersaglio si è spostato in una nuova posizione dopo che il colpo è stato sparato, il server può comunque rilevare correttamente il colpo utilizzando un buffer di stati precedenti per ripristinare lo stato del gioco al momento in cui il colpo è stato sparato. Questo può ridurre significativamente l'effetto del ritardo e rilevare se il colpo avrebbe avuto successo. Tuttavia, ci vuole almeno 1 RTT per gli utenti per ricevere un feedback dalle loro azioni, risultando in difficoltà nella selezione e mira dei bersagli. Inoltre, come nel caso del ritardo aggiunto, l'esperienza per gli utenti non cloud ne risente, ad esempio se un giocatore viene colpito quando già dietro un ostacolo, anche se nello schermo del giocatore cloud non era ancora al riparo, la sua QoE ne risentirà pesantemente, poiché non ci sarà consistenza tra ciò che avviene sullo schermo e come ciò si riflette sugli eventi del gioco.

In conclusione, al momento il cloud gaming rimane una opzione viabile e soddisfacente nel caso in cui si giochi a titoli a giocatore singolo o in cui la latenza tra il giocatore e il server non è parte cruciale delle dinamiche di gioco. Per quanto possa rappresentare il futuro dell'intrattenimento, per il momento e fin quando non si avrà a disposizione una infrastruttura che offra una latenza uguale o quasi a quella degli altri giocatori, resta un mercato di nicchia, ma con molta potenzialità.

5.3.3 Delay compensation tramite estrapolazione

Nel documento "Towards Zero-Latency Video Transmission Through Frame Extrapolation"[52] viene esplorato un approccio per la compensazione del ritardo nella trasmissione video utilizzando l'estrapolazione dei frame. L'obiettivo è ridurre la latenza complessiva, affrontando i limiti tecnologici e fisici che impongono un minimo ritardo incomprimibile nella trasmissione video. La compensazione del ritardo tramite estrapolazione dei frame può essere applicata sia dal lato dell'encoder che dal lato del decoder. In entrambi i casi, l'algoritmo di estrapolazione prevede i frame futuri basandosi su quelli già ricevuti.

Ciò consente di visualizzare un frame previsto mentre il frame reale viene ancora acquisito e trasmesso, riducendo così la latenza percepita.
Nella compensazione dal lato del decoder, il trasmettitore non richiede alcuna modifica rispetto a una pipeline di trasmissione standard. I frame acquisiti vengono compressi e trasmessi al ricevitore, dove arrivano con un certo ritardo. L'algoritmo di estrapolazione eseguito sul decoder utilizza un certo numero di frame decodificati (frame di contesto) per prevedere il frame attuale.
Ad esempio, supponiamo che la latenza *Glass to Glass* (G2G)(il ritardo tra il momento in cui la luce colpisce il sensore della fotocamera e il momento in cui l'immagine viene visualizzata sullo schermo dello spettatore) sia di due frame. Il decoder riceve il frame I_{n-2} (versione compressa di I_{n-2}) quando l'encoder sta acquisendo I_n. L'algoritmo di estrapolazione F utilizza i frame di contesto $\{I_{n-2}, I_{n-3}\}$ per prevedere il frame I_n.
Nella compensazione dal lato dell'encoder, l'algoritmo di estrapolazione viene eseguito sui frame acquisiti prima della compressione e trasmissione. Questo metodo prevede i frame futuri, compensando così la latenza di trasmissione.
Ad esempio, l'algoritmo di estrapolazione F utilizza i frame di contesto $\{I_n, I_{n-1}\}$ per prevedere il frame I_{n+2}. I frame estrapolati vengono compressi, trasmessi, decodificati e visualizzati mentre il frame I_n viene acquisito.
I metodi di estrapolazione considerati includono:

1. **FlowNet-2**: un metodo basato sul movimento che utilizza reti neurali per prevedere i frame successivi.
2. **MCNet**: un metodo basato sui pixel che utilizza reti neurali convoluzionali per la previsione a livello di pixel, catturando la disposizione spaziale e la dinamica temporale delle immagini.
3. **SDCNet**: un approccio ibrido che combina tecniche basate sul movimento e sui pixel.

L'estrapolazione dei frame comporta una perdita di fedeltà, misurata in termini di metriche di qualità come PSNR e SSIM. I risultati preliminari mostrano che, accettando una perdita di qualità, è possibile compensare una latenza tipica di 100 ms con una perdita di circa 8 dB in PSNR utilizzando i migliori algoritmi di estrapolazione.
Gli esperimenti condotti utilizzando il codec HEVC mostrano che l'estrapolazione dal lato dell'encoder e del decoder produce essenzialmente le stesse perdite di qualità per una data quantità di latenza compensata. La qualità dell'estrapolazione dipende significativamente dall'algoritmo utilizzato e dal contenuto video.
La compensazione del ritardo tramite estrapolazione dei frame offre un metodo promettente per ridurre la latenza nella trasmissione video, sebbene comporti una certa perdita di fedeltà. L'ulteriore sviluppo di algoritmi di estrapolazione avanzati potrebbe migliorare l'efficacia di questa tecnica, rendendola applicabile a una gamma più ampia di scenari di trasmissione video in tempo reale.

La tecnica di compensazione del ritardo tramite estrapolazione dei frame offre un metodo promettente per ridurre la latenza nella trasmissione video, sebbene comporti una certa perdita di fedeltà. L'ulteriore sviluppo di algoritmi di estrapolazione avanzati potrebbe migliorare l'efficacia di questa tecnica, rendendola applicabile a una gamma più ampia di scenari di trasmissione video in tempo reale.

La tecnica di compensazione del ritardo tramite estrapolazione dei frame offre un metodo promettente per ridurre la latenza nella trasmissione video, sebbene comporti una certa perdita di fedeltà. L'ulteriore sviluppo di algoritmi di estrapolazione avanzati potrebbe migliorare l'efficacia di questa tecnica, rendendola applicabile a una gamma più ampia di scenari di trasmissione video in tempo reale.

5.3.4 Estrapolazione come alternativa alla riduzione del tasso di codifica

Un ulteriore approfondimento è fornito dal documento "Glass-to-Glass Delay Reduction: Encoding Rate Reduction vs. Video Frame Extrapolation"[85], che confronta due approcci distinti per ridurre la latenza G2G: la riduzione del tasso di codifica e l'estrapolazione dei frame.
La riduzione del tasso di codifica implica una diminuzione del bitrate con cui il video è codificato, riducendo così la quantità di dati da trasmettere per ogni frame e, di conseguenza, il tempo di trasmissione. Questo metodo è noto per la sua semplicità e flessibilità nel controllare la granularità della riduzione della latenza attraverso il fattore di riduzione del tasso di codifica α. Tuttavia, presenta significativi svantaggi, tra cui una notevole perdita di qualità video dovuta agli artefatti di compressione e un limite intrinseco nella riduzione della latenza che non può raggiungere un valore nullo o negativo.
Dall'altro lato, l'estrapolazione dei frame offre un vantaggio significativo in termini di riduzione della latenza, potenzialmente azzerandola o rendendola negativa, specialmente se il ritardo di estrapolazione Δx(tempo che intercorre tra la ricezione di un frame di contesto, i frame già acquisiti, e la generazione del frame previsto) è contenuto. Questo metodo è particolarmente efficace per contenuti video con bassa attività temporale. Tuttavia, richiede un'implementazione più complessa e un maggiore carico computazionale rispetto alla riduzione del tasso di codifica, e può introdurre distorsioni rispetto ai frame originali.
I risultati sperimentali presentati nel documento mostrano che, a parità di riduzione della latenza, l'estrapolazione dei frame tende a mantenere una qualità video superiore rispetto alla riduzione del tasso di codifica. Ad esempio, per un tasso di codifica ridotto a $\alpha = 1/8$, il PSNR risultante è inferiore rispetto a quella ottenuta utilizzando l'estrapolazione con un orizzonte di estrapolazione $h = 5$(numero di frame futuri che l'algoritmo tenta di predire). Le metriche di qualità come PSNR e SSIM evidenziano che l'estrapolazione può mantenere una qualità video accettabile con una maggiore riduzione della latenza rispetto alla riduzione del tasso di codifica. Inoltre, l'estrapolazione è in grado di ridurre la latenza G2G in modo più significativo, specialmente per contenuti con bassa attività temporale.
In conclusione, la tecnica di estrapolazione dei frame video si dimostra più efficace nel ridurre la latenza G2G rispetto alla riduzione del tasso di codifica, offrendo una soluzione promettente per applicazioni che richiedono bassa latenza e alta qualità video. Tuttavia, la riduzione del tempo di estrapolazione e la gestione della qualità rimangono aree chiave per ulteriori miglioramenti.

Capitolo 6

Conclusioni

6.1 Sintesi

In questa tesi, abbiamo presentato e analizzato alcune delle tecnologie e metodologie più recenti nel campo del LS. L'obiettivo principale è stato quello di esaminare le tecniche e le strategie di trasmissione in tempo reale per migliorare la qualità dell'esperienza utente QoE e ridurre la latenza, aspetti chiave nel panorama odierno dello streaming dal vivo.

La nostra analisi è partita dai fondamenti del LS, definendo concetti base e tracciando l'evoluzione storica di questa tecnologia, dall'esperienza degli anni '90 fino ai giorni nostri. Abbiamo iniziato esplorando i protocolli di trasporto tradizionali, come RTP, RTMP e RTSP. Questi protocolli, sebbene robusti e ancora ampiamente utilizzati, presentano alcune limitazioni quando si tratta di soddisfare le crescenti esigenze di qualità e scalabilità richieste dai moderni servizi di LS. In questo contesto, i protocolli più moderni comeHLS, DASH, QUIC e WebRTC offrono vantaggi significativi in termini di adattabilità alle condizioni di rete variabili e miglioramento della QoE.

Uno dei punti chiave del nostro lavoro è stata l'analisi delle tecniche di ABR, che adattano dinamicamente la qualità del video alle condizioni variabili della rete, potenzialmente riducendo il buffering e migliorando la QoE complessiva. Abbiamo approfondito anche l'importanza dell'EC e del MEC per ridurre la latenza e aumentare la scalabilità delle trasmissioni.

Un'altra sezione fondamentale ha riguardato la qualità dell'esperienza utente, dove abbiamo definito il concetto e le metodologie di misurazione, identificando i principali fattori di influenza. Tecniche come l'ottimizzazione basata sul feedback si sono rivelate cruciali per migliorare continuamente la QoE in ambienti di LS.

Particolare attenzione è stata dedicata alla compensazione del ritardo (delay compensation), una sfida fondamentale sia per il gaming online che per il cloud gaming. Abbiamo discusso tecniche specifiche per entrambi i contesti, mettendo in luce l'importanza della riduzione del tasso di codifica e dell'estrapolazione dei frame video. Quest'ultima tecnica, in particolare, si è dimostrata promettente per ridurre la latenza

G2G mantenendo una qualità video accettabile. Abbiamo inoltre evidenziato come l'estrapolazione possa essere un'alternativa valida alla riduzione del bitrate, soprattutto in scenari con bassa attività temporale.

6.2 Sviluppi futuri

I risultati ottenuti con le tecniche di ABR e le innovazioni nel campo delMEC mostrano notevoli potenzialità. Tuttavia, è importante sottolineare che, sebbene abbiamo raggiunto prestazioni significative, esistono ampi margini di miglioramento. Suggeriamo che la ricerca futura si focalizzi sull'ottimizzazione degli algoritmi di ABR per una maggiore efficienza e adattabilità in tempo reale. Inoltre, l'implementazione avanzata del MEC potrebbe ridurre ulteriormente la latenza e migliorare la scalabilità.

Un'altra area di grande interesse è la compensazione del ritardo nel cloud gaming e nel gaming online. Le tecniche di estrapolazione dei frame e la riduzione del tasso di codifica devono essere perfezionate per garantire una bassa latenza senza compromettere la qualità video. È evidente la necessità di algoritmi di estrapolazione più veloci e accurati.

Inoltre, la QoE deve essere continuamente monitorata e migliorata. Le tecniche di ottimizzazione basate sul feedback sono determinanti. La ricerca dovrebbe esplorare nuove metodologie per raccogliere e analizzare il feedback degli utenti in tempo reale, consentendo un adattamento immediato alle condizioni variabili della rete e alle esigenze di una base di utenti eterogenei.

Infine, raccomandiamo che la futura ricerca si impegni a sviluppare nuove soluzioni per migliorare l'efficienza e la robustezza delle tecniche di LS. Un'area promettente è l'integrazione di tecnologie di intelligenza artificiale per ottimizzare in tempo reale la gestione delle risorse di rete e la qualità del flusso video. Inoltre, l'esplorazione di nuove architetture di rete e la sperimentazione con protocolli di trasmissione avanzati potrebbero offrire ulteriori vantaggi in termini di scalabilità e affidabilità.

L'integrazione di queste raccomandazioni nella ricerca futura contribuirà significativamente a migliorare la qualità e l'affidabilità del LS, offrendo esperienze utente superiori e più soddisfacenti.

Capitolo 7

Ringraziamenti

"Non farti spaventare da quei tre anni di università, quei quattro anni saranno i sei più belli della tua vita."

Anonimo

Finalmente anche questo traguardo è stato raggiunto, fatico ancora a crederci. Questi anni accademici sono stati molto intensi, pieni di cambiamenti personali e relazionali, addirittura una pandemia nel mezzo come ciliegina sulla torta. Anni di alti e bassi dove, per fortuna, gli alti sono arrivati verso la fine quasi tutti insieme rendendomi una persona migliore.

Il primo, il più sentito, ringraziamento va alla mia famiglia, senza la quale probabilmente non sarei qui oggi a scrivere questo perché mi è sempre stata accanto nei periodi più difficili dell'università. Periodi in cui ero tentato di abbandonare tutto perché non mi sentivo all'altezza, grazie al loro infinito incoraggiamento e incondizionato supporto mi hanno mostrato come io abbia tutte le carte in regola per poter raggiungere il mio obiettivo con calma e dedizione, cosa che ho dubitato per molti anni.

Un grazie in particolare va a mia nonna Antonia, da sempre fonte di grande affetto e incoraggiamento per il suo *"Nevodo che el xe drio studiare"*. Grazie di tutto nonna.

Un altro grandissimo grazie lo rivolgo a tutti i miei amici, chi più storico chi meno, per avermi tenuto compagnia per tutti questi anni, supportato nei momenti difficili e condiviso quelli più belli.
Ringrazio Alberto, Alessandro (il MALONE), Davide (Ram-Matto), Giogianni (Matita), Giovanni (il ninja di Peraga), Giulio, Marco, Naomi, Sofia e Stefano.

Ultimo, ma non per importanza, desidero ringraziare tutti i professori incontrati durante questi anni universitari. Anche se non sempre è stato facile, il vostro impegno mi ha aiutato a crescere non solo come studente, ma anche come persona. Grazie per aver contribuito alla mia formazione in modi che vanno oltre i libri. In particolare, ringrazio il mio relatore, il Professore Marco Cagnazzo, per la sua disponibilità e il grande supporto offerto sia durante la stesura della tesi sia nel corso dell'insegnamento.

Bibliografia

[1] CISCO. *Cisco Visual Networking Index: Forecast and Trends, 2017–2022*. Rapp. tecn. Cisco public, 2018. URL: https://cloud.report/Resources/Whitepapers/eea79d9b-9fe3-4018-86c6-3d1df813d3b8_white-paper-c11-741490.pdf.

[2] Somayeh Mohammady. *Multiplexing*. eng. IntechOpen, 2019. ISBN: 9781838818685.

[3] Michael Seufert et al. «A Survey on Quality of Experience of HTTP Adaptive Streaming». In: *IEEE Communications Surveys and Tutorials* 17.1 (2015), pp. 469–492. DOI: 10.1109/COMST.2014.2360940.

[4] MediaCollege. *Flash Player Version History*. URL: https://www.mediacollege.com/adobe/flash/player/version/.

[5] Statista. *Number of video game users worldwide from 2019 to 2029*. URL: https://www.statista.com/statistics/748044/number-video-gamers-world/.

[6] Cisco. *Cisco Annual Internet Report (2018–2023) White Paper*. URL: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[7] Sandvine. *Sandvine Global Internet Phenomena Report*. URL: https://www.sandvine.com/hubfs/Sandvine_Redesign_2019/Downloads/2023/reports/Sandvine%20GIPR%202023.pdf.

[8] TwitchTracker. *Twitch Viewers Statistics*. 2024. URL: https://twitchtracker.com/statistics/viewers.

[9] Evi Nemeth et al. *UNIX and Linux System Administration Handbook*. eng. 5th edition. Addison-Wesley Professional, 2018. ISBN: 9780134278308.

[10] Cloudflare. *What is a content delivery network (CDN)? | How do CDNs work?* 2024. URL: https://www.cloudflare.com/learning/cdn/what-is-a-cdn/.

[11] IBM. *Cos'è una CDN (Content Delivery Network)?* 2024. URL: https://www.ibm.com/it-it/topics/content-delivery-networks.

[12] Sam Goundar. *Edge Computing*. Rijeka: IntechOpen, 2023. ISBN: 978-1-83768-862-3. DOI: 10.5772/intechopen.105637. URL: https://doi.org/10.5772/intechopen.105637.

[13] *Intelligent edge computing for cyber physical applications*. eng. Intelligent Data-Centric Systems. San Diego: Elsevier Science & Technology, 2023. ISBN: 0-323-99433-4.

[14] Xiantao Jiang et al. «A Survey on Multi-Access Edge Computing Applied to Video Streaming: Some Research Issues and Challenges». In: *IEEE Communications Surveys and Tutorials* 23.2 (2021), pp. 871–903. ISSN: 1553-877X. DOI: 10.1109/COMST.2021.3065237.

[15] ETSI. *Multi-access Edge Computing (MEC)*. 2024. URL: https://www.etsi.org/technologies/multi-access-edge-computing.

[16] Dario Sabella, Alex Reznik e Rui Frazao. *Multi-Access Edge Computing in Action*. Set. 2019, p. 230. ISBN: 9780429508752. DOI: 10.1201/9780429056499.

[17] Dario Sabella. *Multi-access edge computing : software development at the network edge*. eng. Textbooks in telecommunication engineering. Cham, Switzerland: Springer, 2022. ISBN: 3-030-79618-3.

[18] *Mobile Edge Computing: A key technology towards 5G*. ETSI White Paper, 2015. URL: https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp11_mec_a_key_technology_towards_5g.pdf.

[19] Henning Schulzrinne et al. *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550. Lug. 2003. DOI: 10.17487/RFC3550. URL: https://www.rfc-editor.org/info/rfc3550.

[20] A. C. M. Fong e S. C. (Siu Cheung) Hui. *Multimedia engineering a practical guide for internet implementation*. eng. RSP series in industrial control, computers and communications. Chichester, England ; Wiley, 2006. ISBN: 0-470-03085-2.

[21] R. Zurawski. *The Industrial Information Technology Handbook*. Industrial Electronics. CRC Press, 2018. ISBN: 9781420036336. URL: https://books.google.it/books?id=MwMDUBKZ3wwC.

[22] C. Perkins. *RTP: Audio and Video for the Internet*. Kaleidoscope Series. Addison-Wesley, 2003. ISBN: 9780672322495. URL: https://books.google.it/books?id=OM7YJAy9_m8C.

[23] Hans W. Barz e Gregory A. Bassett. *Multimedia networks : protocols, design, and applications*. eng. Hoboken: John Wiley & Sons Inc., 2016. ISBN: 1-119-09017-2.

[24] A. Farrel. *The Internet and Its Protocols: A Comparative Approach*. ISSN. Elsevier Science, 2004. ISBN: 9780080518879. URL: https://books.google.it/books?id=LtBegQowqFsC.

[25] Veriscope. *Adobe RTMP Specification*. 2024. URL: https://rtmp.veriskope.com/docs/spec/.

[26] Parikshit Juluri, Venkatesh Tamarapalli e Deep Medhi. «Measurement of Quality of Experience of Video-on-Demand Services: A Survey». In: *IEEE Communications Surveys & Tutorials* 18 (feb. 2015), pp. 1–1. DOI: 10.1109/COMST.2015.2401424.

[27] Abdelhak Bentaleb et al. *Toward One-Second Latency: Evolution of Live Media Streaming*. 2023. arXiv: 2310.03256 [cs.NI].

[28] Anup Rao, Rob Lanphier e Henning Schulzrinne. *Real Time Streaming Protocol (RTSP)*. RFC 2326. Apr. 1998. DOI: 10.17487/RFC2326. URL: https://www.rfc-editor.org/info/rfc2326.

[29] Abid Yaqoob, Ting Bi e Gabriel-Miro Muntean. «A Survey on Adaptive 360° Video Streaming: Solutions, Challenges and Opportunities». In: *IEEE Communications Surveys and Tutorials* 22.4 (2020), pp. 2801–2838. DOI: 10.1109/COMST.2020.3006999.

[30] Markus Rupp e Michal Ries. «Video streaming test bed for UMTS network». In: ().

[31] IETF. *Real-Time Streaming Protocol Version 2.0*. 2016. URL: https://www.rfc-editor.org/rfc/rfc7826.

[32] Roger Pantos e William May. *HTTP Live Streaming*. RFC 8216. Ago. 2017. DOI: 10.17487/RFC8216. URL: https://www.rfc-editor.org/info/rfc8216.

[33] Roger Pantos. *HTTP Live Streaming 2nd Edition*. Internet-Draft draft-pantos-hls-rfc8216bis-15. Work in Progress. Internet Engineering Task Force, mag. 2024. 110 pp. URL: https://datatracker.ietf.org/doc/draft-pantos-hls-rfc8216bis/15/.

[34] Mukaddim. Pathan, Ramesh Kumar Sitaraman e Dom. Robinson. *Advanced content delivery, streaming, and cloud services*. eng. 1st edition. Wiley series on parallel and distributed computing. Hoboken, New Jersey: Wiley, 2016. ISBN: 1-118-90964-X.

[35] Apple. *HTTP Live Streaming*. 2024. URL: https://developer.apple.com/documentation/http-live-streaming.

[36] Iraj Sodagar. «The MPEG-DASH Standard for Multimedia Streaming Over the Internet». In: *IEEE MultiMedia* 18.4 (2011), pp. 62–67. DOI: 10.1109/MMUL.2011.71.

[37] Cloudflare. *What is MPEG-DASH? | HLS vs. DASH*. 2024. URL: https://www.cloudflare.com/learning/video/what-is-mpeg-dash/.

[38] Jonathan Kua, Grenville Armitage e Philip Branch. «A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming Over HTTP». In: *IEEE Communications Surveys and Tutorials* 19.3 (2017), pp. 1842–1866. ISSN: 1553-877X. DOI: 10.1109/COMST.2017.2685630.

[39] Google H. Alvestrand. *RFC 8835 Transports for WebRTC*. 2021. URL: https://www.rfc-editor.org/rfc/rfc8835.html.

[40] Google. *WebRTC Overview*. 2024. URL: https://webrtc.org/getting-started/overview.

[41] mdn. *WebRTC API*. 2021. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API.

[42] Roar Hagen et al. *Internet Low Bit Rate Codec (iLBC)*. RFC 3951. Dic. 2004. DOI: 10.17487/RFC3951. URL: https://www.rfc-editor.org/info/rfc3951.

[43] Jean-Marc Valin, Koen Vos e Timothy B. Terriberry. *Definition of the Opus Audio Codec*. RFC 6716. Set. 2012. DOI: 10.17487/RFC6716. URL: https://www.rfc-editor.org/info/rfc6716.

[44] Louay Bassbouss et al. «Streamlining WebRTC and DASH for near-real-time media delivery». In: *SMPTE 2022 Media Technology Summit*. SMPTE. 2022, pp. 1–17.

[45] Cloudflare. *WebRTC live streaming to unlimited viewers, with sub-second latency*. 2024. URL: https://blog.cloudflare.com/webrtc-whip-whep-cloudflare-stream/.

[46] Jim Roskind. *QUIC: Design Document and Specification Rationale*. 2013. URL: https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/edit.

[47] Jana Iyengar e Martin Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. Mag. 2021. DOI: 10.17487/RFC9000. URL: https://www.rfc-editor.org/info/rfc9000.

[48] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Ago. 2018. DOI: 10.17487/RFC8446. URL: https://www.rfc-editor.org/info/rfc8446.

[49] Michael Driscoll. *GitHub - syncsynchalt/illustrated-quic: The Illustrated QUIC Connection: Every byte explained*. URL: https://github.com/syncsynchalt/illustrated-quic.

[50] Tianchi Huang et al. «Learning Tailored Adaptive Bitrate Algorithms to Heterogeneous Network Conditions: A Domain-Specific Priors and Meta-Reinforcement Learning Approach». In: *IEEE Journal on Selected Areas in Communications* 40.8 (2022), pp. 2485–2503. DOI: 10.1109/JSAC.2022.3180804.

[51] Yunlong Li et al. «Fleet: Improving Quality of Experience for Low-Latency Live Video Streaming». In: *IEEE Transactions on Circuits and Systems for Video Technology* 33.9 (2023), pp. 5242–5256. DOI: 10.1109/TCSVT.2023.3243901.

[52] Melan Vijayaratnam et al. «Towards Zero-Latency Video Transmission Through Frame Extrapolation». In: *2022 IEEE International Conference on Image Processing (ICIP)*. 2022, pp. 2122–2126. DOI: 10.1109/ICIP46576.2022.9897958.

[53] DASH Industry Forum. *ThroughputRule.js*. URL: https://github.com/Dash-Industry-Forum/dash.js/blob/development/src/streaming/rules/abr/ThroughputRule.js.

[54] DASH Industry Forum. *ThroughputController.js*. URL: https://github.com/Dash-Industry-Forum/dash.js/blob/development/src/streaming/controllers/ThroughputController.js#L115.

[55] Kevin Spiteri, Rahul Urgaonkar e Ramesh K. Sitaraman. «BOLA: Near-Optimal Bitrate Adaptation for Online Videos». In: *IEEE/ACM Transactions on Networking* 28.4 (2020), pp. 1698–1711. DOI: 10.1109/TNET.2020.2996964.

[56] Zhi Li et al. «Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale». In: *IEEE Journal on Selected Areas in Communications* 32.4 (2014), pp. 719–733. DOI: 10.1109/JSAC.2014.140405.

[57] mdn. *Web video codec guide*. 2024. URL: https://developer.mozilla.org/en-US/docs/Web/Media/Formats/Video_codecs.

[58] Cloudflare. *How does live stream encoding work? | Video encoding*. 2024. URL: https://www.cloudflare.com/learning/video/live-stream-encoding/.

[59] Federico Gianno. *Streaming in Depth: A Technical Exploration of Twitch's Protocols*. 2023. URL: https://federicogianno.medium.com/streaming-secrets-a-comprehensive-look-at-twitch-tvs-technical-infrastructure-bbe3d5d81736.

[60] YUESHI SHEN KAI HAYASHI ANGELO PAXINOS. *Low Latency, High Reach: Creating an Unparalleled Live Video Streaming Network at Twitch*. 2023. URL: https://blog.twitch.tv/en/2021/10/25/low-latency-high-reach-creating-an-unparalleled-live-video-streaming-network-at-twitch/.

[61] Twitch. *Twitch Engineering: An Introduction and Overview*. 2015. URL: https://blog.twitch.tv/en/2015/12/18/twitch-engineering-an-introduction-and-overview-a23917b71a25/.

[62] T. Wiegand et al. «Overview of the H.264/AVC video coding standard». In: *IEEE Transactions on Circuits and Systems for Video Technology* 13.7 (2003), pp. 560–576. DOI: 10.1109/TCSVT.2003.815165.

[63] Debargha Mukherjee et al. «A Technical Overview of VP9–the Latest Open-Source Video Codec». In: *SMPTE Motion Imaging Journal* 124 (feb. 2015), pp. 44–54. DOI: 10.5594/j18499.

[64] Gary J. Sullivan et al. «Overview of the High Efficiency Video Coding (HEVC) Standard». In: *IEEE Transactions on Circuits and Systems for Video Technology* 22.12 (2012), pp. 1649–1668. DOI: 10.1109/TCSVT.2012.2221191.

[65] Gary J. Sullivan et al. «Overview of the High Efficiency Video Coding (HEVC) Standard». In: *IEEE Transactions on Circuits and Systems for Video Technology* 22.12 (2012), pp. 1649–1668. DOI: 10.1109/TCSVT.2012.2221191.

[66] Jingning Han et al. «A Technical Overview of AV1». eng. In: *Proceedings of the IEEE* 109.9 (2021), pp. 1435–1462. ISSN: 0018-9219.

[67] Miroslav Uhrina et al. «Performance Comparison of VVC, AV1, HEVC and AVC for High Resolutions». In: (feb. 2024). DOI: 10.20944/preprints202402.0869.v1.

[68] Twitch Blog. *Introducing the Enhanced Broadcasting Beta*. URL: https://blog.twitch.tv/en/2024/01/08/introducing-the-enhanced-broadcasting-beta/.

[69] *ITU-T Recommendation P.10: Vocabulary for performance and quality of service*. URL: https://www.itu.int/rec/T-REC-P.10-201711-I/en.

[70] Kjell Brunnström et al. *Qualinet White Paper on Definitions of Quality of Experience*. Mar. 2013, pp. –.

[71] *BT.500 : Methodologies for the subjective assessment of the quality of television images*. URL: https://www.itu.int/rec/R-REC-BT.500-15-202305-I/en.

[72] *P.910 : Subjective video quality assessment methods for multimedia applications*. URL: https://www.itu.int/rec/T-REC-P.910-202310-I/en.

[73] Zhou Wang et al. «Image quality assessment: from error visibility to structural similarity». In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.

[74] Saeed Shafiee Sabet. *The influence of delay on cloud gaming quality of experience*. eng. T-Labs Series in Telecommunication Services. Cham, Switzerland: Springer, 2022. ISBN: 9783030998691.

[75] Mingzhe Yang et al. «Survey on QoE Assessment Approach for Network Service». In: *IEEE Access* 6 (2018), pp. 48374–48390. DOI: 10.1109/ACCESS.2018.2867253.

[76] Sabina Barakovic, Jasmina Barakovic Husic e Himzo Bajric. «QoE dimensions and QoE measurement of NGN services». In: gen. 2010.

[77] Lea Skorin-Kapov e Martín Varela. «A multi-dimensional view of QoE: the ARCU model». In: *2012 Proceedings of the 35th International Convention MIPRO*. 2012, pp. 662–666.

[78] R. Stankiewicz e A. Jajszczyk. «A survey of QoE assurance in converged networks». In: *Computer Networks* 55.7 (2011). Recent Advances in Network Convergence, pp. 1459–1473. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2011.02.004. URL: https://www.sciencedirect.com/science/article/pii/S138912861100051X.

[79] Fatima Laiche, Asma Ben Letaifa e Taoufik Aguili. «QoE Influence Factors (IFs) classification Survey focusing on User Behavior/Engagement metrics». In: *2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. 2020, pp. 143–146. DOI: 10.1109/WETICE49692.2020.00036.

[80] Tianchi Huang et al. *Optimizing Adaptive Video Streaming with Human Feedback*. 2023. arXiv: 2308.04132 [cs.MM].

[81] Takato Motoo et al. «Client-Side Network Delay Compensation for Online Shooting Games». In: *IEEE Access* 9 (2021), pp. 125678–125690. DOI: 10.1109/ACCESS.2021.3111180.

[82] Kyungmin Lee et al. «Outatime: Using Speculation to Enable Low-Latency Continuous Interaction for Mobile Cloud Gaming». In: *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 151–165. ISBN: 9781450334945. DOI: 10.1145/2742647.2742656. URL: https://doi.org/10.1145/2742647.2742656.

[83] Zhi Li et al. «Lag Compensation for First-Person Shooter Games in Cloud Gaming». In: *Autonomous Control for a Reliable Internet of Services: Methods, Models, Approaches, Techniques, Algorithms, and Tools*. A cura di Ivan Ganchev, R. D. van der Mei e Hans van den Berg. Cham: Springer International Publishing, 2018, pp. 104–127. ISBN: 978-3-319-90415-3. DOI: 10.1007/978-3-319-90415-3_5. URL: https://doi.org/10.1007/978-3-319-90415-3_5.

[84] Jiawei Sun e Mark Claypool. «Evaluating Streaming and Latency Compensation in a Cloud-based Game». In: 2019. URL: https://api.semanticscholar.org/CorpusID:197676898.

[85] Hind Kanj et al. «Glass-to-Glass Delay Reduction: Encoding Rate Reduction vs. Video Frame Extrapolation». In: *2023 IEEE 25th International Workshop on Multimedia Signal Processing (MMSP)*. 2023, pp. 1–6. DOI: 10.1109/MMSP59012.2023.10337718.