UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA DELL'INFORMAZIONE

"ARCHITETTURE E PROTOCOLLI DI ULTIMA GENERAZIONE PER IL LIVE STREAMING EFFICIENTE"

Relatore: Prof. Nicola Laurenti

Laureando: Luca Furlan

ANNO ACCADEMICO 2021 – 2022

Data di laurea 25/11/2022

INDICE:

Capitolo 1

INTRODUZIONE

Negli ultimi anni i servizi di delivery video hanno subito una forte impennata e in particolare lo streaming live ha rappresentato un aspetto trainante per l'industria grazie ad una rapida e decisa crescita. Un rapporto, ad esempio, mostra che "i video in diretta sono cresciuti del 93%, con un tempo medio di visione di 26,4 minuti per sessione" [1]. Sempre più persone accedono a contenuti in diretta, 7 su 10 dichiarano di connettersi in uno streaming live quotidianamente. [2]
Inoltre, la componente di live streaming è diventata sempre più centrale comportando uno spostamento dalla fruizione di contenuto broadcast televisivo ad un contenuto multimediale, in diretta e non, attraverso la rete, come sostenuto da una ricerca di GWI: "le generazioni più giovani (Gen Z e Millennials) passano più tempo in streaming, mentre le persone più anziane (Boomers e Gen X) guardano di più la TV broadcast." [3]

Piattaforme di live streaming come Youtube Live, Facebook Live e Twitch hanno visto crescere di molto la loro popolarità guadagnando numerosi utenti in poco tempo, servizi basati sullo streaming in diretta sono nati per distribuire contenuti prima ad appannaggio solamente di servizi broadcast, come il campionato di calcio della Serie A.

A causa anche della pandemia il mondo si è spostato velocemente in una direzione di interazione digitale, sfruttando i video in diretta come parte integrante di questa interazione. Grazie al live streaming è stato possibile raggiungere diffusamente gli utenti con contenuti particolarmente adatti al live streaming, tra i quali possiamo inserire:

- Eventi virtuali
- Educazione a distanza
- Coperture sportive
- Concerti in streaming
- Riunioni d'affari
- Vendite video
- Marketing
- Servizi religiosi

- Riunioni politiche
- Trasmissioni creative

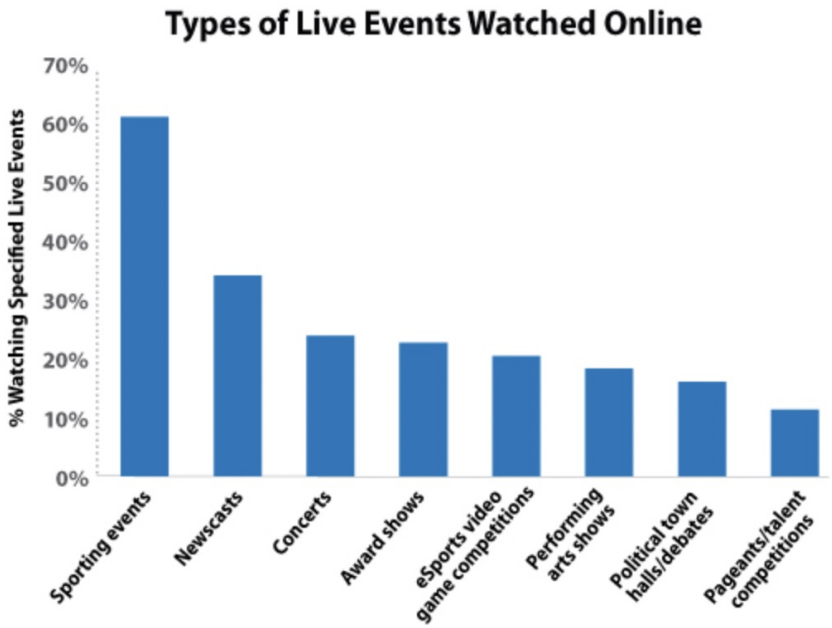


Figura 1 - Tipologia di eventi Live guardati online

Ad esempio, una ricerca mostra come due terzi degli operatori di marketing che si rivolgono ai consumatori utilizzano il live streaming per il marketing. [4]
Non solo gli utenti hanno potuto beneficiare di questa tecnologia, essa ha potuto aiutare anche in maniera utile e efficiente settori quali:

- Sorveglianza remota
- Guida remota
- Prevenzione incendi

Il live streaming è diventato dunque una parte integrante dell'esperienza online di molte persone, e questo grazie anche alla creazione secondo necessità di metodi e strategie che sapessero concentrarsi precisamente in questo ambito, grazie alla creazione di protocolli specifici e architetture di rete adatte allo scopo. Per questo nuove metriche che si adattassero ai problemi del live streaming sono nate allo scopo di poter analizzare e migliorare l'esperienza utente.

Nonostante la sua larga diffusione in un tempo così relativamente breve, il live streaming presenta ancora numerose sfide che gli si pongono davanti. La riduzione di latenza rimane

ancora una problematica cardine nell'inseguimento di una trasmissione in tempo reale, e la qualità percepita dall'utente finale ha ancora margini di miglioramento.

In questa tesi verranno analizzati i protocolli e le architetture per il live streaming efficiente, da ciò che è in uso oggi fino a ciò che sarà in arrivo nel prossimo futuro. Partendo con l'enunciare le metriche di misurazione per un live streaming di qualità, sia oggettive che soggettive, verrà poi introdotta la strategia adottata diffusamente per il miglioramento attivo delle suddette metriche. Successivamente verranno introdotti i principali protocolli e ne verranno approfondite struttura e funzionamento. In seguito, si affronteranno confronti ricavati da alcuni studi per poi considerare nuovi possibili metodi di miglioramento degli stessi grazie a ricerche recenti nell'ambito. Per finire analizzeremo le architetture di rete che permettono la diffusione del live streaming.

Capitolo 2

METRICHE PER IL LIVE STREAMING

I principali parametri che possono essere usati per misurare le prestazioni vengono definiti metriche, e costituiscono i fattori principali attraverso cui è possibile valutare la qualità di un flusso multimediale.
Le metriche principali possono essere divise in due insiemi [5], l'insieme delle metriche legate alla *Quality of Service* (QoS) e quelle metriche legate alla *Quality of Experience* (QoE), quest'ultimo suddivisibile a sua volta in fattori legati alla percezione dello spettatore e fattori tecnici.

2.1 QoE

Con quality of experience si intende il risultato a livello di esperienza che riceve ogni spettatore partecipante al live streaming, in particolare l'insieme di fattori che veicolano un certo stato di insoddisfazione nella visione del flusso video e possono portare eventualmente anche all'abbandono della visione.
Possiamo identificare due macro gruppi:

- Fattori di percezione
- Fattori tecnici

2.1.1 Fattori di percezione:

In questo gruppo sono inseriti tutti i termini collegati che incidono a livello di sensazioni e di percezioni dall'utente finale dell'applicazione e sono disaccoppiati dallo sviluppo tecnico del live streaming. Ad esempio, diverse ragioni tecniche possono contribuire a introdurre un ritardo iniziale, ma l'utente finale percepirà solamente il tempo di attesa derivante.
All'interno di questa sezione possiamo porre:

- Mean Opinion Score
- Tempi di attesa

    - Ritardo iniziale
    - Statistica delle durate degli stalli

- Adattamento video
    - Frequenza di switching
    - Ampiezza
    - Tempo in ogni Layer

- Qualità video
    - Risoluzione spaziale
    - Scalabilità temporale
    - Qualità d'immagine

- Fattori di contesto
    - Dispositivo
    - Contenuto
    - Utilizzo

Nello specifico in questa sezione verranno fornite descrizioni qualitative delle metriche di cui sopra, con esempi di calcolo per il ritardo iniziale, la statistica di durata degli stalli e l'adattamento video.

2.1.1.1 Mean Opinion Score

Il Mean Opinion Score (MOS) è una metrica utilizzata per rappresentare la qualità complessiva di un flusso multimediale attraverso la media aritmetica di tutti i singoli valori su una scala predefinita che un soggetto assegna alla sua opinione sulle prestazioni di un sistema di qualità.
Tali valutazioni sono solitamente raccolte in un test soggettivo di valutazione della qualità, in genere nell'intervallo 1-5, dove 1 è la qualità percepita più bassa e 5 è la qualità percepita più alta. Siano R le valutazioni individuali per un dato stimolo da parte di N soggetti abbiamo:

$$MOS = \frac{\sum_{n=1}^{N} R_n}{N}$$

Il MOS per essere calcolato richiede il setup di un esperimento che coinvolga diversi individui, per questo motivo è generalmente considerato costoso in termini di tempo e risorse.

2.1.1.2 Ritardo iniziale:

Ogni trasferimento dati attraverso una rete porta con sé un ritardo insito nella struttura e nella propagazione del dato in essa. In uno streaming multimediale non vi è eccezione e una certa quantità di dati deve essere trasferita dalla sorgente a destinazione prima che la riproduzione possa iniziare, questo crea un ritardo percepibile dall'utente finale. Il valore pratico del ritardo iniziale minimo raggiungibile dipende quindi dalla velocità di trasmissione disponibile e dalle impostazioni del codificatore. Di solito, la riproduzione video viene ritardata più di quanto tecnicamente necessario per dare modo al buffer di riproduzione di riempirsi con una quantità maggiore di tempo di riproduzione video nel ricevitore al momento iniziale. Il buffer di riproduzione è uno strumento efficiente utilizzato per affrontare le variazioni di throughput a breve termine. Tuttavia, la quantità di tempo di riproduzione inizialmente bufferizzato deve essere bilanciata tra la lunghezza effettiva del ritardo corrispondente (più tempo di riproduzione bufferizzato = ritardo iniziale più lungo) e il rischio di esaurimento del buffer, cioè di stallo (più tempo di riproduzione bufferizzato = maggiore robustezza alle variazioni di throughput a breve termine).

Il riferimento [6] osserva come il ritardo iniziale è preferito allo stallo nella riproduzione da circa il 90% degli utenti, poiché a differenza del ritardo iniziale previsto, che è l'attesa prima dell'avvio dello streaming ed è ben noto nell'uso quotidiano delle applicazioni video, lo stallo invoca un'improvvisa e inaspettata interruzione del servizio.

Lo stesso studio mostra inoltre come un ritardo iniziale fino a 16 secondi riduca la qualità percepita solo marginalmente.

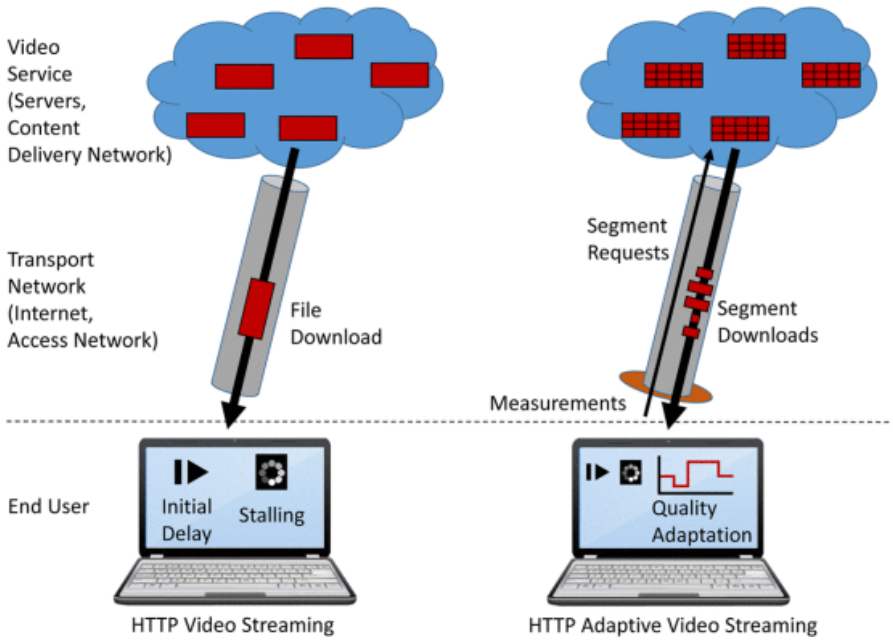


Figura 2 - L'utente finale non è consapevole delle influenze del servizio o della rete, ma percepisce solo i ritardi iniziali, lo stallo e gli adattamenti della qualità.

In altri studi, riferimenti [7] e [8], che si concentrano sull'esperienza degli spettatori all'interno di servizi basati sui contenuti generati dagli utenti, si osserva come il ritardo iniziale abbia un impatto maggiore, essendo gli utenti meno tolleranti nell'attesa durante la loro navigazione tra la grande mole di contenuti.
Ne consegue che per le implementazioni di servizi video, come per qualsiasi altro servizio, i ritardi iniziali dovrebbero essere tenuti possibilmente brevi, ma in questo caso i ritardi iniziali non rappresentano un problema di prestazioni importante.

Come riportato nell'esperimento del paper al riferimento [19], la metrica di ritardo iniziale può essere calcolata nel seguente modo:

$$ST = T_{loading} + TTNC + T_{decode} \quad [ms]$$

In questa formula, $T_{loading}$ si riferisce al tempo necessario alla preparazione del video, questo include i ritardi dovuti per esempio alla eventuale gestione della protezione del contenuto multimediale. T_{decode} indica il tempo impiegato per decodificare e renderizzare i chunk scaricati, mentre con TTNC si indica il tempo impiegato per il download del minimo numero di chunk per iniziare la riproduzione.

2.1.1.3 Statistica delle durate degli stalli:

L'interruzione durante la riproduzione di un contenuto multimediale è definita stallo. Essa è causata principalmente dall'esaurimento dei dati nel buffer per via di un bitrate del video superiore al throughput dello streaming video. Lo stallo durerà finchè nel buffer dell'applicazione non vi saranno immagazzinati nuovamente abbastanza dati per riprendere la riproduzione. Il tempo di attesa anche qui gioca un ruolo fondamentale, essendo necessario bilanciare la durata dell'interruzione con la grandezza del segmento bufferizzato (lunghe attese = più tempo di buffer immagazzinato = meno possibilità di interruzioni future).
Nel riferimento [5] l'autore conclude che lo stallo impatta molto severamente la qualità percepita di un live streaming e che andrebbe evitato il più possibile, sia in frequenza che in durata.

La statistica delle durate degli stalli può essere misurata in diversi modi:

- Il conteggio degli stalli è il numero di volte in cui la riproduzione si è bloccata.
- La durata dello stallo è il tempo totale in cui la riproduzione è rimasta in stallo.

- La frequenza di stallo è la frequenza con cui si verificano gli eventi di stallo (come il conteggio degli stalli / minuti di visione).
- La percentuale di stallo è la percentuale del tempo di visione dello spettatore in cui la riproduzione è stata bloccata (durata dello stallo / tempo di visione).
- Il rapporto di stallo è il rapporto tra la durata dello stallo e la durata effettiva del video riprodotto (durata dello stallo / durata della riproduzione).

In riferimento all'experimental setup [19] preso in considerazione nella sezione precedente, il rapporto di stallo viene definito nel seguente modo: intuitivamente, il tempo di stallo viene stimato tenendo conto dei segmenti video che sono stati scaricati e la parte del video che avrebbe dovuto essere riprodotta finora. Sia B_i l'occupazione del buffer video in secondi appena prima che il segmento venisse scaricato. Il tempo di stallo tra due tempi di download di segmenti consecutivi, ET_i e ET_{i-1}, è rappresentato da b_i. Sia j l'indice del segmento dopo che la riproduzione è ripresa dall'ultimo evento di stallo, e CTS il numero minimo di segmenti richiesti dal buffer per avviare la riproduzione. All'inizio, $j = CTS$ e $b_k = 0$ per $k \leq CTS$, poiché il tempo di attesa prima dell'avvio del video è considerato come tempo di avvio per definizione. Per ogni successivo segmento i, B_i è calcolato come segue:

$$B_i = max\left((i - 1 - j + CTS) \times L - (ET_i - ET_j), 0\right).$$

Qui, $(i - 1 - j + CTS) \times L$ rappresenta il contenuto video che è stato scaricato, e $(ET_i - ET_j)$ rappresenta il video totale che dovrebbe essere stato riprodotto dall'inizio dell'ultima riproduzione. Se $B_i > 0$, allora $b_i = 0$ e si passa al segmento successivo. In caso contrario, si verifica uno stallo, calcolato come segue:

$$b_i = (ET_i - ET_j) - (i - 1 - j + CTS) \times L.$$

In questo caso, la riproduzione del video inizierà dopo aver scaricato CTS segmenti. Pertanto, il valore di j viene impostato su $i + CTS - 1$ e il parametro b_k per il segmento $k \in \{i + 1, i + CTS - 1\}$ è impostato come $ET_k - ET_{k-1}$. I restanti valori di b_i possono essere ottenuti in modo analogo. Il rapporto di stallo può essere calcolato come segue:

$$\widehat{RR} = \frac{\sum_{k=1}^{N} b_k}{N \times L + \sum_{k=1}^{N} b_k},$$

dove N denota il numero di segmenti osservati durante la sessione mentre L è la durata del segmento in secondi. La sommatoria $\sum_{k=1}^{N} b_k$ rappresenta il tempo totale di stallo.

2.1.1.4 Adattamento video:

Un live streaming può utilizzare tecniche per adattarsi a diversi throughput e alle loro fluttuazioni. Per rendere possibile l'adattamento, è necessario cambiare il paradigma dello streaming, in modo che il client, che può misurare le condizioni attuali della rete ai margini della stessa, controlli quale velocità di trasmissione dei dati è adatta alle condizioni attuali. Impiegando queste tecniche vengono però introdotte nuove possibili problematiche avvertite dall'utente finale. Ad esempio, lo streaming video si adatta alla banda stimata in download passando da un bitrate maggiore ad un bitrate inferiore (*switching*), questo causa un degradamento della qualità video (un bitrate inferiore porta meno informazioni di un bitrate maggiore a parità delle altre condizioni) ed eventuali salti di qualità la cui frequenza ed ampiezza possono causare insoddisfazione nello spettatore. Nei riferimenti [9] e [10] gli studi mostrano come l'adattamento della qualità può ridurre efficacemente lo stallo dell'80% quando il throughput diminuisce ed è responsabile di un migliore utilizzo del throughput disponibile quando questo aumenta. Inoltre [10] conferma che è meglio controllare la qualità che subire effetti incontrollati come lo stallo. Gli autori confrontano la degradazione dello streaming video dovuta alla perdita di pacchetti con quella dovuta alla riduzione della risoluzione. I risultati oggettivi vengono mappati sulla qualità percepita soggettivamente e scoprono che l'impatto della degradazione incontrollata (cioè la perdita di pacchetti) sulla QoE è molto più grave dell'impatto di una riduzione controllata del throughput dovuta alla risoluzione.

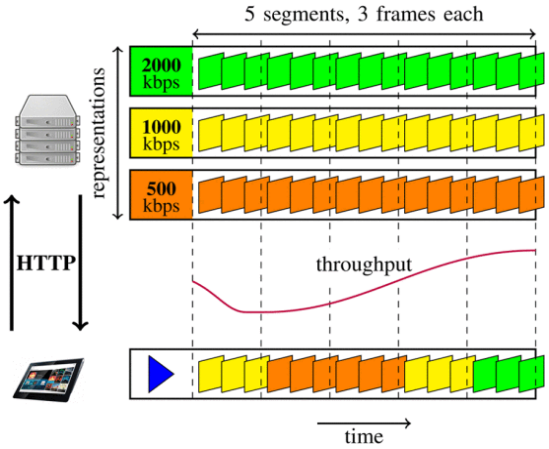


Figura 3 - Adattamento video in uno streaming HTTP

Riprendendo il riferimento [19] la frequenza di switching viene calcolata a partire dal numero totale di volte che il bitrate stimato del segmento è cambiato tra due segmenti consecutivi. Sia N il numero di segmenti e D la durata in minuti di ogni segmento, viene stimata la frequenza di switching al minuto (S) nel modo seguente:

$$\hat{S} = \frac{\sum_{l=2}^{N} I(\widehat{Q_l} \neq \widehat{Q_{l-1}})}{N \times D} \qquad [min^{-1}]$$

dove *I* è la funzione indicatore che è uguale a uno se i segmenti consecutivi non hanno lo stesso bitrate, altrimenti è zero.

2.1.1.5 Qualità video:

La qualità video ricevuta dall'utente finale in un live streaming può essere analizzata attraverso la sua risoluzione e temporizzazione, oltre che dal suo bitrate intrinseco.
La risoluzione di un live streaming rappresenta il numero di pixel con cui l'immagine è costruita (ad esempio 1920x1080 Full HD, 3840x2160 4K) e viene decisa in fase di trasmissione, essa può variare in modi spiegati al paragrafo precedente attraverso meccaniche di adattamento video. Una diminuzione della risoluzione porta ad un abbassamento della QoE [11].
La temporizzazione in un flusso video indica il frame rate, ovvero la frequenza con cui i fotogrammi sono presentati allo spettatore (Frames per second = FPS), l'alta velocità con cui questi frames si susseguono in un video fornisce l'illusione del movimento. La variabilità della frequenza può far percepire un effetto scattoso nel video, degradando la fluidità. Una diminuzione del frame rate porta ad una diminuzione della QoE come mostrato in [12], ma questo cambiamento nella QoE è fortemente dipendente dalla tipologia di contenuto in movimento [13].

2.1.1.6 Fattori di contesto:

I fattori di contesto rappresentano elementi indipendenti dalla trasmissione e ricezione di un flusso live streaming, e che quindi non possono essere modificati dall'infrastruttura o dalla strategia di trasmissione scelta a monte. Essi sono legati all'utente finale e sono: la scelta del dispositivo di riproduzione (dispositivi diversi con diverse performance possono riprodurre in maniera differente la stessa trasmissione), il contenuto guardato e l'utilizzo dell'applicazione di riproduzione.

2.1.2 Fattori tecnici:

I fattori tecnici che guidano la percezione degli utenti finali giocano un ruolo nella QoE percepita a destinazione.
In questa sezione possiamo trovare:

- Server side:
    - Codec video
    - Grandezza segmenti

- Client side:
    - Riproduttore video
    - Adattamento
    - Buffer video

- Logica di adattamento
    - Monitoraggio della QoS
    - Stima del bitrate
    - Riempimento del buffer

2.1.2.1 Server side:

Il versante server si concentra nell'identificare gli elementi che influiscono nel flusso streaming decisi prima che il flusso venga consegnato al destinatario. Durante la codifica delle immagini originali in immagini codificate adatte ad essere trasportate dal protocollo scelto, il server utilizza determinati codec (in maniera dipendente dal protocollo di streaming scelto) affinché l'immagine possa essere trasportata con efficacia e riprodotta una volta a destinazione.

I segmenti video sono comuni nelle tecnologie di streaming basate su HTTP come vedremo nei capitoli successivi, ma possono essere utilizzati anche in altre tipologie di streaming. Un segmento video (o chunk) è un frammento di informazioni video formato da un insieme di fotogrammi video, e combinati insieme, questi segmenti costituiscono un intero video. Nello streaming, i segmenti video hanno dimensioni diverse. La comprensione delle dimensioni dei segmenti video in uno streaming può aiutare a determinare la dimensione del segmento più efficiente e del ritardo sorgente-destinazione. Infatti, come investigato in [9] e mostrato nella Figura 4, la grandezza del segmento influisce direttamente nel numero di stalli in uno streaming video.

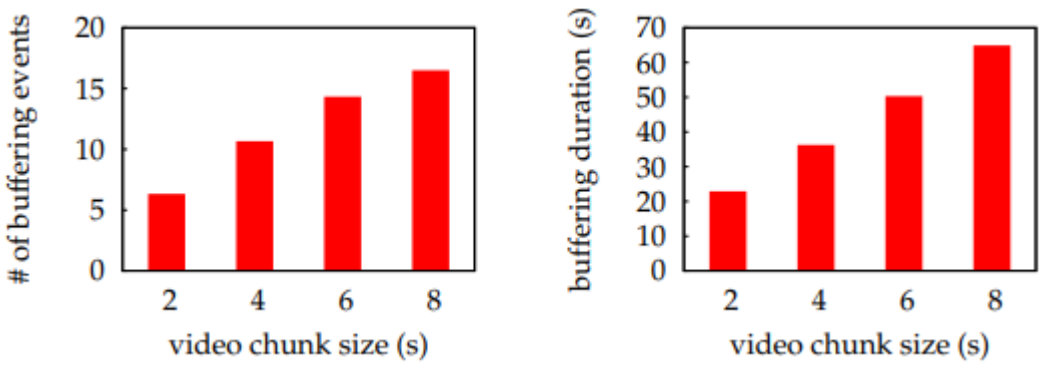


Figura 4 - Numero e durata degli stalli in relazione alla grandezza del segmento video

Dunque, i segmenti devono essere sufficientemente corti per consentire una reazione rapida ai cambiamenti delle condizioni di rete, ma allo stesso tempo devono essere sufficientemente lunghi per consentire un'elevata efficienza di codifica del codificatore video sorgente. Chiaramente, questi due requisiti costituiscono un problema di ottimizzazione che deve essere preso in considerazione dal lato server durante la preparazione dei contenuti.

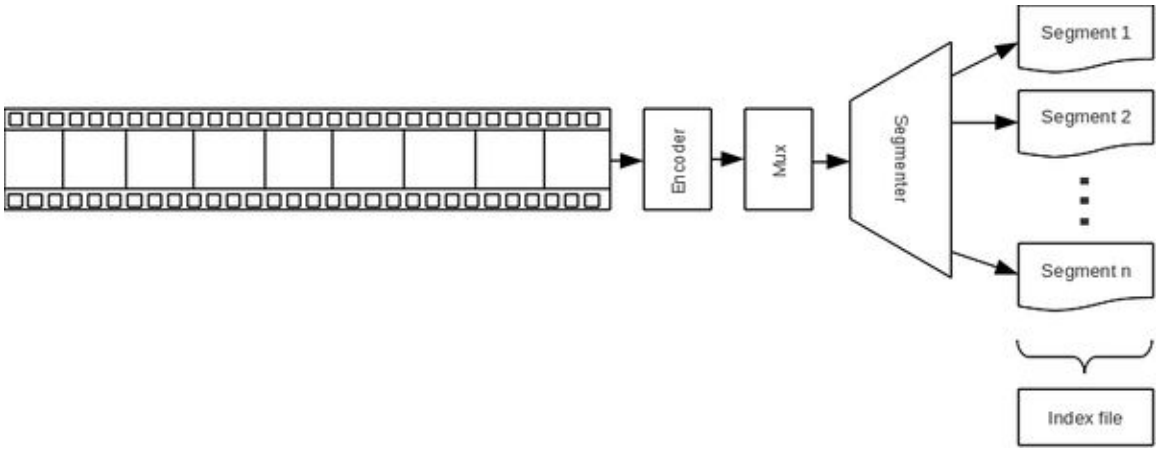


Figura 5 - Preparazione dei segmenti video lato server

2.1.2.2 Client side:

Nel lato client si identificano i fattori collegati alla riproduzione del flusso video a destinazione, e le decisioni più importanti sono quali segmenti scaricare, quando iniziare il download e come gestire il buffer video del ricevitore.

L'adattamento dello streaming viene deciso da un algoritmo decisionale implementato al client che sfrutta le informazioni più rilevanti, come la stima sul throughput istantaneo, per scegliere un criterio decisionale volto alla massimizzazione della QoE.

Il buffer video del client rappresenta un luogo virtuale in cui immagazzinare i segmenti video da riprodurre. Questo permetterà, nel caso in cui ci sia un blocco del flusso di alcuni secondi, la continuazione della visione senza causare particolari problemi.

2.1.2.3 Logica di adattamento:

A seconda del caso d'uso e dello scenario, si possono utilizzare diverse strategie di adattamento per adattarsi alla diversa velocità di trasmissione disponibile. Esse fanno riferimento principalmente al livello di riempimento del buffer video, alla stima del throughput istantaneo al ricevitore e al bitrate medio di ogni segmento calcolato come il bitrate medio di tutti i segmenti video richiesti da un utente.

2.2 QoS:

La QoS è una misura più oggettiva. Si riferisce alle prestazioni di una rete di trasmissione e comprende fattori importanti come i tempi di uptime, la probabilità di interruzione, i tassi di errore, il throughput e la latenza. La QoS si concentra maggiormente sul processo che avviene tra l'IP e l'applicazione di streaming, piuttosto che sugli utenti finali stessi. La QoS è fondamentale per i fornitori di contenuti ad alto traffico, come gli eventi sportivi e musicali in diretta streaming, poiché una QoS scadente influisce notevolmente sulla QoE.

Secondo il distributore CDN Akamai [14], ben il 50% degli spettatori abbandona uno streaming se impiega più di 5 secondi per caricarsi. Il 76% degli spettatori abbandona anche un servizio se lo streaming viene bufferizzato più volte.

Nei fattori rilevanti per la QoS possiamo trovare:

- Latenza
    - latenza di buffer
    - congestione

- Perdita pacchetti/Corruzione/Out of order
    - ritrasmissione pacchetti
    - riordinamento

- Jitter o Packet Pacing

2.2.1 Latenza:

La latenza dello stream è il ritardo tra l'acquisizione di un fotogramma video durante un evento e la sua visualizzazione da parte dello spettatore. Il passaggio di pacchetti di dati da un luogo all'altro richiede tempo, quindi la latenza si accumula in ogni fase del flusso di lavoro dello streaming. Il termine latenza *end-to-end* viene utilizzato per rappresentare la differenza di tempo totale tra la sorgente e lo spettatore. Altri termini, come "latenza di acquisizione" o "latenza del player", tengono conto solo del ritardo introdotto in una fase specifica del flusso di lavoro dello streaming.

Le fasi della latenza comprendono:

- Telecamera che elabora l'immagine
- Codificatori che effettuano la transcodifica del contenuto
- Tempo di trasmissione del video allo spettatore
- Buffering
- Il dispositivo di riproduzione multimediale decodifica e visualizza il video

I dati grezzi devono essere codificati, un processo che comporta la compressione e la formattazione dei file multimediali per il trasferimento in rete. Questi file vengono poi inviati a un server di streaming, che consente di transcodificarli (decodificarli, modificarli in base alle dimensioni o alla risoluzione e ricodificarli) e/o trasmutarli (passare a un protocollo diverso per la consegna). Infine, possono passare attraverso una rete di distribuzione dei contenuti (CDN) per raggiungere vari dispositivi di riproduzione e venire caricati nei buffer video.

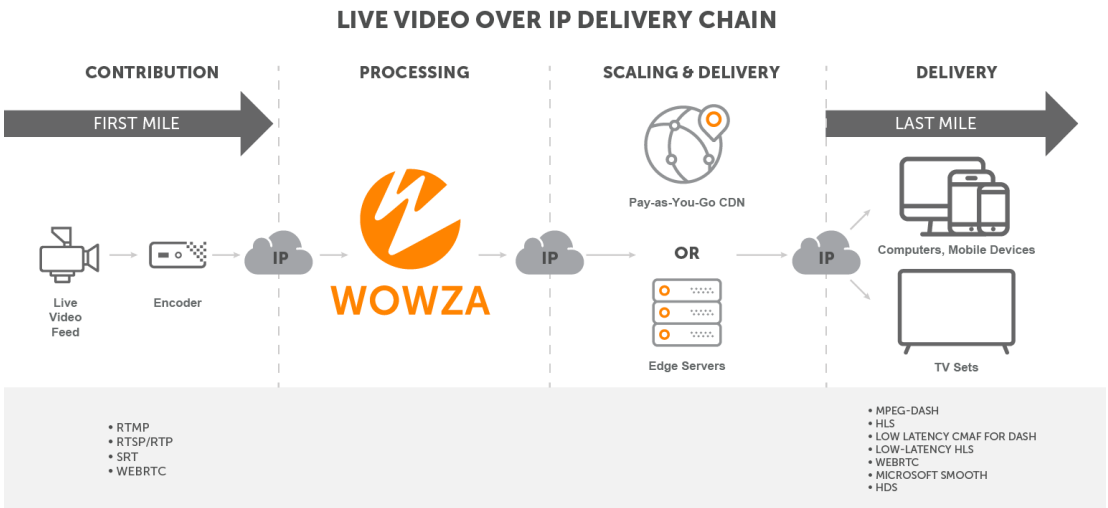


Figura 6 - Sequenza di fasi per la consegna video in un Live streaming

Possiamo riassumere gli step in due parti principali:

- Il tempo di avvio per la riproduzione

- lo streaming video sullo schermo dello spettatore

Entrambi sono importanti per l'esperienza dello spettatore. Purtroppo, in alcuni casi, la riduzione di un componente della latenza può aumentare l'altro.

La latenza viene misurata da unità di tempo abbastanza piccole da distinguere tra due eventi consecutivi.

2.2.2 Perdita pacchetti/Corruzione/Out of order:

In qualsiasi ambiente di rete, i dati vengono inviati e ricevuti attraverso la rete in piccole unità chiamate pacchetti. La perdita pacchetti è un fenomeno che si presenta nelle situazioni in cui i pacchetti dati durante uno streaming non raggiungono la destinazione prevista. La perdita di pacchetti viene spesso misurata come rapporto tra il numero di pacchetti persi e il numero totale di pacchetti inviati. Quando i dati devono essere consegnati in modo affidabile, i pacchetti vengono solitamente ritrasmessi, ma questo può aumentare la latenza della rete. La perdita di pacchetti può verificarsi per diversi motivi, dalle limitazioni del throughput ai firewall del dispositivo che bloccano i trasferimenti di dati, ma la causa più comune è la congestione della rete.

La perdita di pacchetti durante lo streaming video è un problema perché influisce negativamente sull'esperienza di visione. Ad esempio, se tutti i pacchetti di dati di una particolare sezione del video vengono persi, possono verificarsi problemi come immagini pixelate, lacune nei contenuti video e persino un'interruzione totale della riproduzione. Per questo motivo le emittenti utilizzano diversi metodi per ridurre la perdita di pacchetti durante lo streaming video, come tecniche di *Forward Error Correction* (FEC) o *Automatic Repeat Request* (ARQ).

La corruzione di pacchetto avviene quando il pacchetto arrivato a destinazione differisce da quello originale, in questo caso se l’errore non viene rilevato la riproduzione video presenterà delle distorsioni, mentre se la corruzione viene rilevata verrà richiesta una ritrasmissione.

Out-of-order rappresenta una condizione in cui i pacchetti sono arrivati non secondo l’ordine con cui devono essere riprodotti, causando del ritardo per il riordinamento. La consegna out-

of-order può essere causata da pacchetti che seguono percorsi multipli attraverso una rete, da procedure di ritrasmissione di livello inferiore (come la ARQ) o da percorsi di elaborazione paralleli all'interno di apparecchiature di rete che non sono progettate per garantire il mantenimento dell'ordine dei pacchetti. Una delle funzioni del TCP è quella di prevenire la consegna fuori ordine dei dati, riassemblando i pacchetti in ordine o richiedendo la ritrasmissione dei pacchetti fuori ordine.

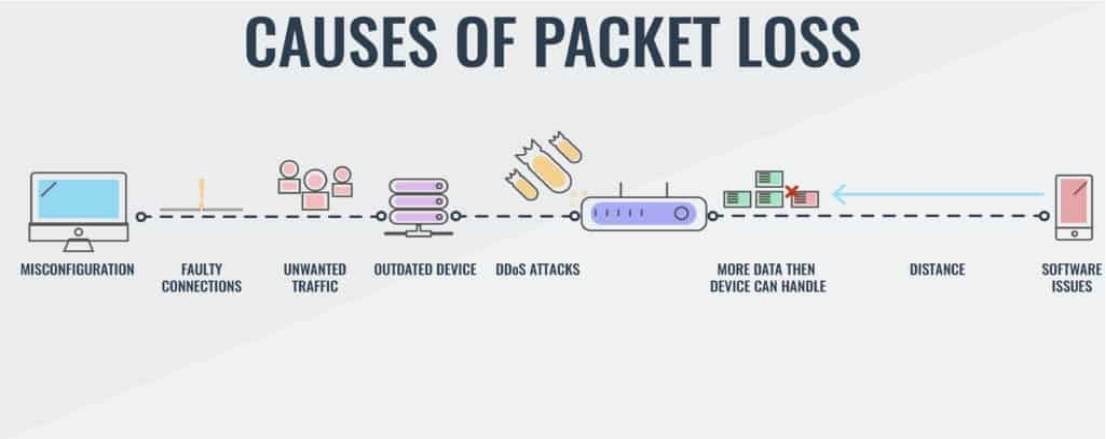


Figura 7 - Principali cause per la perdita di pacchetti

2.2.3 Jitter e Packet pacing:

Jitter è una parola inglese che significa più o meno "tremolio", ed è un indice indiretto di stabilità della trasmissione del segnale.
[20] definisce il jitter come il valore assoluto della differenza tra il ritardo di inoltro di due pacchetti consecutivi ricevuti appartenenti allo stesso flusso.

Ad esempio, quando due pacchetti (pacchetti A e B) vengono inviati attraverso una rete:
Il pacchetto A impiega 15 ms per attraversare la rete.
Il pacchetto B impiega 18 ms per attraversare la rete.
La differenza di latenza tra i due pacchetti della coppia è di 3 ms. Jitter = | 15 - 18 | = 3 ms.

Il calcolo del jitter richiede la misurazione di quattro parametri:

- Tempo di trasmissione del primo pacchetto della coppia ($T_T(n)$)
- Tempo di ricezione del primo pacchetto della coppia ($T_R(n)$)
- Tempo di trasmissione del secondo pacchetto della coppia ($T_T(n+1)$)
- Tempo di ricezione del secondo pacchetto della coppia ($T_R(n+1)$)

Il jitter J può essere espresso come:

$$J(n) = |(T_R(n) - T_T(n)) - (T_R(n+1) - T_T(n+1))| \quad [ms]$$

Il jitter è sempre espresso come un numero positivo, quindi si utilizza il valore assoluto della differenza. Il jitter medio è definito come il valore medio del jitter di coppie di pacchetti consecutivi. Se la latenza è costante e ogni pacchetto subisce lo stesso ritardo, il jitter sarebbe pari a 0 (poiché la differenza di latenza da un pacchetto all'altro non cambia).

Il packet pacing rappresenta il ritmo di arrivo dei pacchetti, che in una connessione ottimale non deve superare un certo vincolo. Pacchetti che arrivano in “raffiche” separate possono causare perdita di pacchetti e incrementare il jitter.

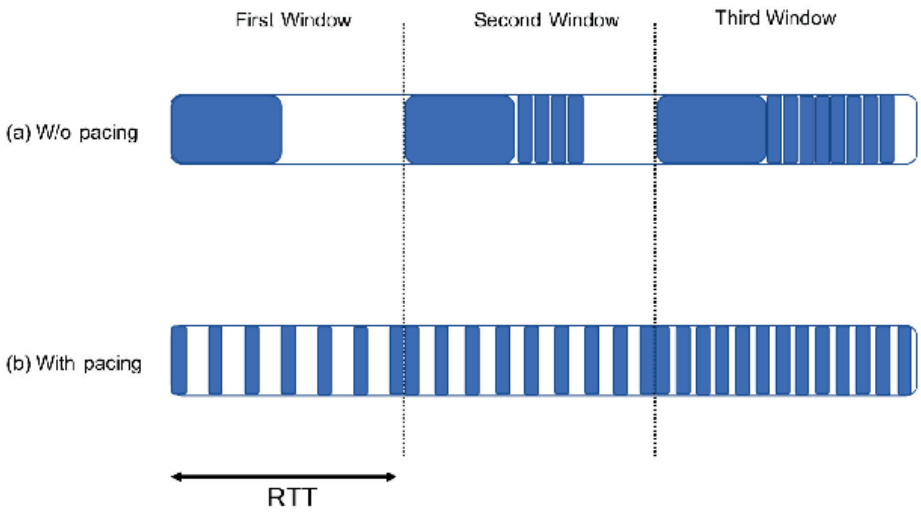


Figura 8 - Rappresentazione del packet pacing

2.3 Adaptive BitRate Streaming

Lo streaming a bitrate adattivo non è una metrica, ma viene introdotto nella seguente sezione essendo uno dei fattori principali che lavorano al miglioramento della QoE per lo spettatore. *Adaptive BitRate* (ABR) è una tecnica utilizzata in ambito streaming, tipicamente su reti HTTP, che permette di adattare la qualità di una trasmissione multimediale, in termini di bitrate medio al throughput del destinatario, allo scopo di ridurre al minimo gli stalli e il buffering. L'obiettivo principale degli algoritmi ABR è quello di prevenire l'esaurimento del buffer di riproduzione del client, massimizzando al contempo la qualità dell'esperienza percepita (QoE) dell'utente grazie all'adattamento alle condizioni di rete che cambiano dinamicamente.

Lo streaming a bitrate adattativo inizia nella fase di codifica video. La codifica è il processo in cui i video non compressi vengono convertiti in una forma che può essere memorizzata e utilizzata su molti dispositivi. Affinché lo streaming a bitrate adattativo funzioni, è necessario creare diversi file video che supportino bitrate differenti.

Dopo la codifica, il video viene segmentato in file più piccoli, della durata di pochi secondi. Nella maggior parte delle configurazioni di streaming, i video vengono trasmessi in una serie di segmenti, piuttosto che un intero file video inviato tutto in una volta. Il processo di segmentazione è particolarmente importante perché, senza di esso, i lettori video dovrebbero scaricare l'intero file video prima di poter iniziare la riproduzione del contenuto.

Inoltre, i segmenti sono importanti per lo streaming a bitrate adattivo, perché il processo di regolazione si attiva alla fine di un segmento video. Se la connessione di uno spettatore non è in grado di scaricare il video abbastanza velocemente da consentire lo streaming senza buffering, il lettore video passerà a un file più piccolo una volta terminato il segmento.

Quando si avvia la riproduzione di un video, molti lettori video iniziano a richiedere il file con il bitrate più basso disponibile. Se il lettore determina che il client è in grado di gestire un file a bitrate più elevato, selezionerà file a bitrate più elevato fino a trovare quello più alto che il client è in grado di gestire. Se il file selezionato è quello ideale per la connessione, il lettore continuerà a richiedere segmenti a quella velocità, a meno che le condizioni non cambino. Questa è la cosiddetta "scala" di bitrate o codifica adattiva. Il lettore sale la scala quando la connessione ha un throughput sufficiente per ospitare video a bitrate più elevati e scende la scala quando questa diminuisce.

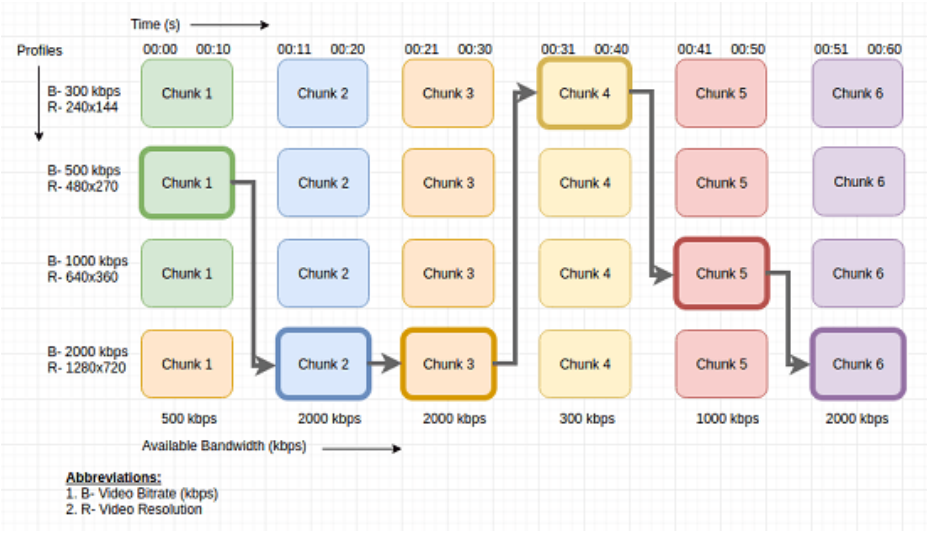


Figura 9 - Rappresentazione della selezione di segmenti con bitrate diverso

Una recente ricerca [15] descrive una serie di schemi e tecniche di adattamento del bitrate, in questo caso per il protocollo DASH su HTTP. Gli autori hanno classificato gli schemi in tre categorie principali: lato client, lato server e tramite approcci in rete. Hanno fornito una rassegna generale dei metodi di misurazione del traffico video e una serie di studi di caratterizzazione per noti fornitori di streaming commerciale come Netflix, YouTube e Akamai.

La maggior parte delle soluzioni di *HTTP Adaptive Streaming* (HAS) all'avanguardia integra la logica di adattamento del bitrate esclusivamente all'interno del client HAS, in quanto consente al client di selezionare un livello di bitrate in modo indipendente ed evita la necessità di disporre di dispositivi intelligenti all'interno dell'infrastruttura di rete. Questo rappresenta un motivo fondamentale per cui le soluzioni HAS vengono utilizzate negli scenari OTT. Tuttavia, un client può utilizzare anche il feedback riportato da un server o dalla rete nell'adattamento del bitrate per migliorare la QoE complessiva. Lo studio inoltre classifica gli schemi in quattro categorie principali: basati su client, basati su server, assistiti dalla rete e ibridi. La classificazione si basa sulla posizione della logica di adattamento del bitrate all'interno del sistema e sulle entità coinvolte.

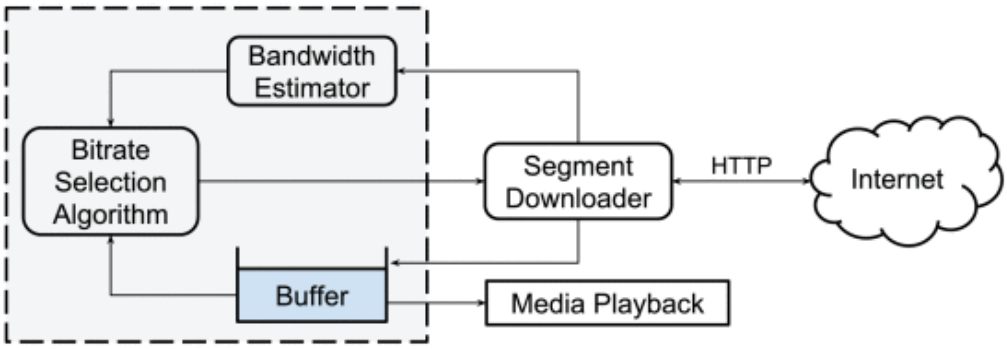


Figura 10 - Schema di una soluzione HAS

Capitolo 3

PROTOCOLLI PER IL LIVE STREAMING

I protocolli per il live streaming si concentrano nel descrivere regole e metodi standard per sezionare file video in segmenti più piccoli, in modo tale da poterli consegnare velocemente all'utente finale per, infine, riassemblarli in unico flusso video. Essi ricadono principalmente nel livello di applicazione, ma protocolli differenti si possono basare su differenti sotto-protocolli di trasporto (come ad esempio TCP/UDP).

Ogni protocollo per il live streaming può basarsi su un diverso protocollo di trasporto e utilizzare strategie differenti per la consegna del flusso video al destinatario, questo ha portato alla creazione di una varietà di protocolli diversi che potremo differenziare in:

- Protocolli tradizionali basati su un utilizzo classico di TCP/UDP
    - RTMP
    - RTSP/RTP

- Protocolli adattivi basati su HTTP
    - HLS
    - LL-HLS
    - DASH
    - LL-DASH

- Protocolli di ultima generazione basati su versioni evolute di TCP/UDP
    - SRT
    - WebRTC
    - RIST

Per poter sfruttare i vantaggi di protocolli differenti, essi possono essere utilizzati in maniera combinata nella stessa trasmissione dati e il seguente diagramma illustra le quattro fasi distinte nella catena di distribuzione dei flussi video live, la cui prima sezione viene spesso chiamata "*first mile*" e i protocolli utilizzati sono talvolta chiamati "*ingest protocols*", mentre

l'ultima, in cui i pacchetti passano dai server di rete al dispositivo dell'utente finale, viene chiamata "*last mile*" e i protocolli detti *"egress protocols"*.

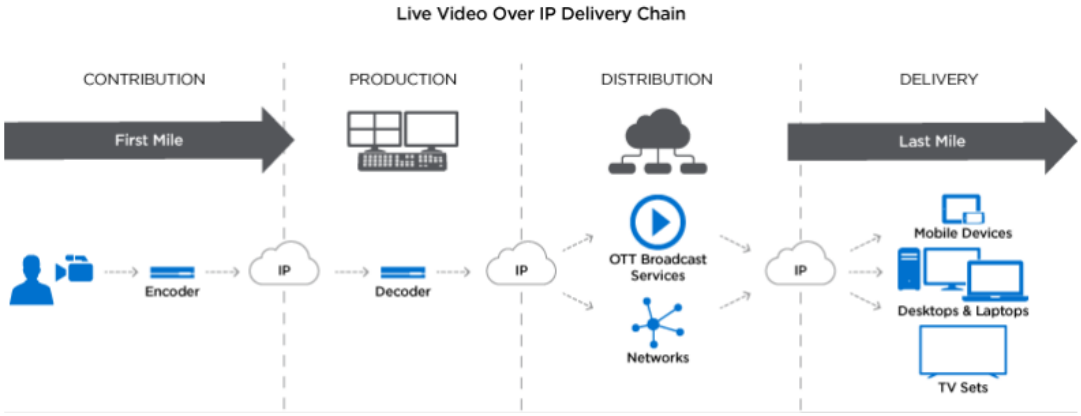


Figura 11 - Catena di distribuzione di un flusso in streaming

3.1 Protocolli tradizionali basati su un utilizzo classico di TCP/UDP:

3.1.1 RTMP

Il protocollo RTMP (ovvero *Real-Time Messaging Protocol*) è stato creato originariamente nel 2002 come protocollo proprietario da Macromedia (successivamente acquisita da Adobe) per essere utilizzato nel trasferimento audio e video tra un server di streaming e l'applicativo proprietario "Flash Player". Successivamente Adobe ha fornito le specifiche del protocollo per uso pubblico nel 2008.

L'RTMP rappresenta il primo protocollo diventato uno standard de-facto grazie all'enorme diffusione di Flash come distribuzione di contenuti video (più del 90% dei browser internet lo supportava nei primi anni 2000)

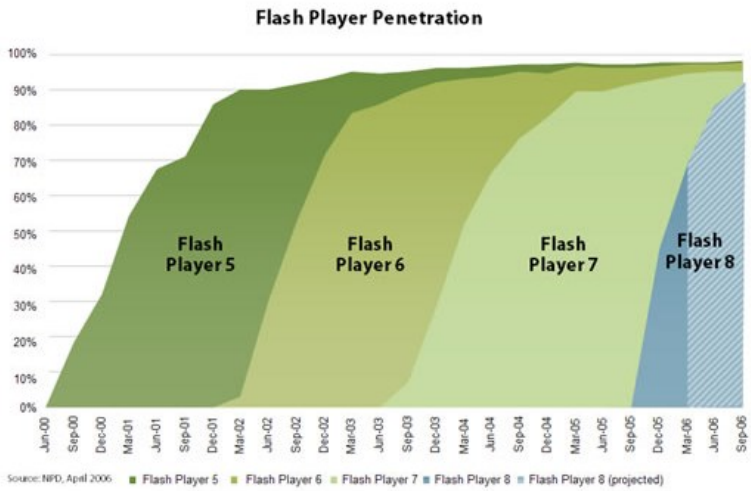


Figura 12 - Diffusione di Flash Player nei primi anni 2000

Adobe definisce il RTMP come segue:

"*The Real-Time Messaging Protocol (RTMP) was designed for high-performance transmission of audio, video, and data between Adobe Flash Platform technologies, including Adobe Flash Player and Adobe AIR. RTMP is now available as an open specification to create products and technology that enable the delivery of video, audio, and data in the open AMF, SWF, FLV, and F4V formats compatible with Adobe Flash Player.*"

Nonostante la sua larga diffusione, il veloce abbandono di Flash a causa delle criticità di sicurezza e la nascita di altri protocolli più moderni basati su HTTP ha reso presto obsoleto il protocollo RTMP per la trasmissione dati ai player video, ma rimane ancora utilizzato in maniera importante come *ingest protocol* per la trasmissione dati da sorgente ai server di elaborazione nel *first mile*.
Ora che i dispositivi che supportano Flash (e di conseguenza RTMP) sono meno numerosi che mai, la stessa Adobe incoraggia i distributori di contenuti a "*migrate any existing Flash content to… new open formats.*".

3.1.1.1 Protocollo base:

RTMP funziona basandosi su un protocollo inferiore nella pila protocollare, il protocollo di trasporto Transmission Control Protocol (TCP) e utilizza il numero di porta 1935 per impostazione predefinita.

3.1.1.2 Struttura:

Il protocollo RTMP sfrutta la suddivisione dei grossi file sorgente in file più piccoli, che rappresentano una frazione del file originario, chiamati *chunk*. A ogni chunk creato è associato un ID univoco, chiamato chunk stream ID. I chunk vengono trasmessi sulla rete. Durante la trasmissione, ogni chunk deve essere inviato per intero prima del chunk successivo. All'estremità del ricevitore, i chunk vengono assemblati in messaggi basati sul chunk stream ID.
Il chunking consente di suddividere i messaggi di grandi dimensioni del protocollo di livello superiore in messaggi più piccoli, ad esempio per evitare che i messaggi di grandi dimensioni a bassa priorità (come il video) blocchino i messaggi più piccoli ad alta priorità (come l'audio o il controllo).

Il chunking consente inoltre di inviare messaggi di piccole dimensioni con un minore overhead, poiché lo header del chunk contiene una rappresentazione compressa delle informazioni che altrimenti dovrebbero essere incluse nel messaggio stesso.
La dimensione del chunk è configurabile. Può essere impostata tramite un messaggio di controllo definito *Set Chunk Size*. Le dimensioni maggiori dei chunk riducono l'uso della CPU, ma comportano anche scritture più grandi che possono ritardare altri contenuti su connessioni a basso throughput di banda. La dimensione dei chunk viene mantenuta in modo indipendente per ogni direzione.

3.1.1.3 Funzionamento:

Una connessione RTMP inizia con un handshake. L'handshake è diverso dal resto del protocollo: consiste in tre pacchetti di dimensioni statiche, anziché in pacchetti di dimensioni variabili con header. Il client (l'endpoint che ha avviato la connessione) e il server inviano ciascuno gli stessi tre pacchetti. Per comodità espositiva, questi chunk saranno denominati C0, C1 e C2 se inviati dal client; S0, S1 e S2 se inviati dal server.

I pacchetti C0/S0 sono lunghi 1 byte e sono utilizzati per stabilire quale versione del RTMP utilizzare. I pacchetti C1/S1 invece hanno una lunghezza di 1536 byte e stabiliscono un timestamp di riferimento a cui tutti i pacchetti successivi dello stream faranno riferimento. Infine i pacchetti C2/S2 sono simili a quelli precedenti, sempre lunghi 1536 byte, e contengono due tipi di timestamp: il primo è il timestamp inviato dal peer in S1 (per C2) o C1 (per S2), il secondo è il timestamp in cui è stato letto il pacchetto precedente (S1 o C1) inviato dal peer.

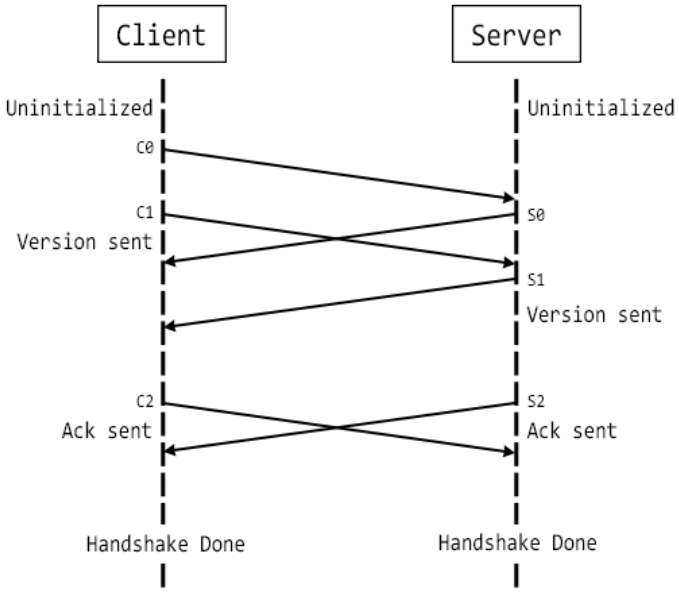


Figura 13 - Sequenza di Handshake

Di seguito vengono descritti gli stati menzionati nel diagramma di handshake:

- *Uninitialized:* In questa fase viene inviata la versione del protocollo. Sia il client che il server non sono inizializzati. Il client invia la versione del protocollo nel pacchetto C0. Se il server supporta la versione, invia S0 e S1 in risposta. In caso contrario, il server risponde con un'azione appropriata. In RTMP, questa azione consiste nel terminare la connessione.
- *Version sent*: Sia il client che il server si trovano nello stato version sent dopo lo stato uninitialized. Il client è in attesa del pacchetto S1 e il server è in attesa del pacchetto C1. Alla ricezione dei pacchetti attesi, il client invia il pacchetto C2 e il server invia il pacchetto S2. Lo stato diventa quindi Ack Sent.
- *Ack Sent*: Il client e il server attendono rispettivamente S2 e C2.
- *Handshake done*: Il client e il server si scambiano i messaggi.

Nella fase successiva allo handshake, il server e il client iniziano a configurare la connessione per lo scambio dati attraverso uno scambio di comandi.

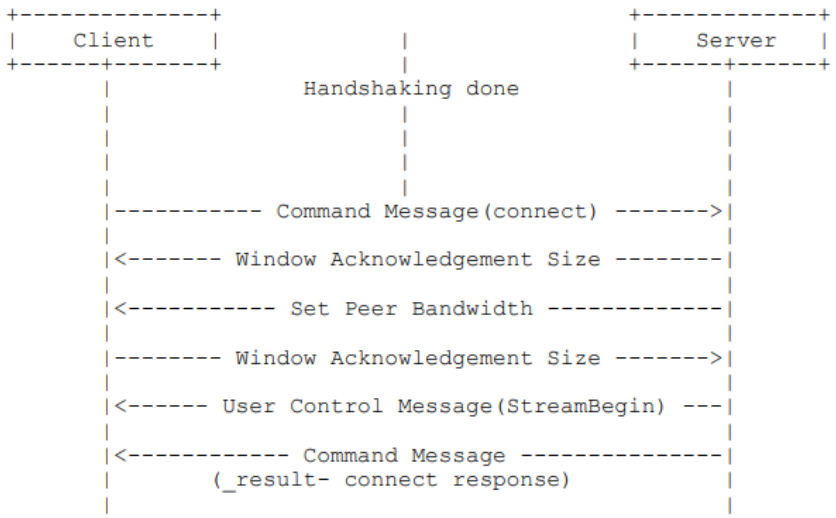


Figura 14 - Sequenza dei comandi di connessione

1. Il client invia il comando connect al server per richiedere la connessione con l'istanza dell'applicazione del server.
2. Dopo aver ricevuto il comando di connessione, il server invia al client il messaggio di protocollo “Window Acknowledgement Size” (questo messaggio serve per informare il peer della dimensione della finestra da utilizzare tra l'invio di *acknowledgment*). Anche il server si connette all'applicazione indicata nel comando di connessione.

3. Il server invia al client il messaggio di protocollo "Set Peer Bandwidth" (questo messaggio serve per limitare il throughput di banda in uscita del proprio peer. Il peer che riceve questo messaggio limita il throughput di banda in uscita limitando la quantità di dati inviati ma riconosciuti alla dimensione della finestra indicata in questo messaggio).
4. Il client invia il messaggio di protocollo “Window Acknowledgement Size” al server dopo aver elaborato il messaggio di protocollo “Set Peer Bandwidth”.
5. Il server invia un altro messaggio di protocollo del tipo User Control (StreamBegin) al client.
6. Il server invia il messaggio di comando risultante per informare il client sullo stato della connessione (successo/fallimento). Il comando specifica l'ID della transazione e anche le proprietà, ad esempio la versione del Flash Media Server.

Una volta che la configurazione della connessione è stata completata, può essere avviato lo stream del contenuto.

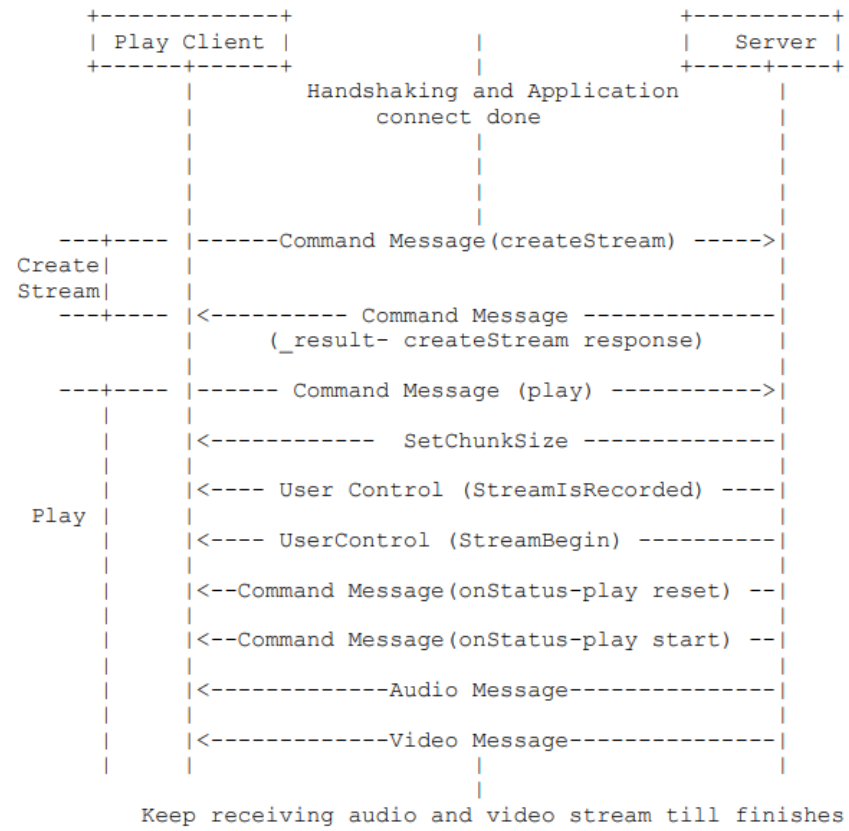


Figura 15 - Sequenza dei comandi di avvio stream

Il flusso dei messaggi sarà:

1. Il client invia il comando play dopo aver ricevuto con successo il risultato del comando createStream dal server.

2. Alla ricezione del comando play, il server invia un messaggio di protocollo per impostare la dimensione del chunk.
3. Il server invia un altro messaggio di protocollo (controllo utente) specificando l'evento "StreamIsRecorded" e l'ID di stream in tale messaggio. Il messaggio riporta il tipo di evento nei primi 2 byte e l'ID di stream negli ultimi 4 byte.
4. Il server invia un altro messaggio di protocollo (controllo utente) che specifica l'evento 'StreamBegin', per indicare l'inizio dello streaming al client.
5. Il server invia un messaggio di comando onStatus NetStream.Play.Start e NetStream.Play.Reset se il comando di riproduzione inviato dal client ha successo. NetStream.Play.Reset viene inviato dal server solo se il comando di riproduzione inviato dal client ha impostato il flag di reset. Se il flusso da riprodurre non viene trovato, il server invia il messaggio onStatus NetStream.Play.StreamNotFound.

Successivamente, il server invia i dati audio e video che il client riproduce.

Se il client invia il comando play2, lo stream può passare a un flusso di bitrate diverso senza modificare la timeline del contenuto riprodotto. Il server mantiene più file per tutte le velocità di trasmissione supportate che il client può richiedere con play2.

3.1.2 RTSP/RTP

Una variante del RTMP è il protocollo RTSP/RTP (*Real-Time Streaming Protocol/Real Time Protocol*) è formato da un protocollo di presentazione (RTSP) che permette all'utente finale di controllare il flusso remoto, mentre RTP funge da protocollo di trasporto. Il suo utilizzo principale è la creazione e il controllo di sessioni multimediali, come TV e film, tra endpoint. RTSP richiede un server dedicato per lo streaming e non supporta la crittografia dei contenuti o la ritrasmissione dei pacchetti persi, poiché si basa sul protocollo RTP insieme a RTCP per la consegna dei flussi multimediali.

I dispositivi Android e iOS non dispongono di lettori compatibili con RTSP, per cui questo protocollo viene utilizzato raramente per la riproduzione. Detto questo, RTSP rimane uno standard in molte architetture di video sorveglianza e di televisioni a circuito chiuso (CCTV) poichè il supporto RTSP è ancora diffusissimo nelle telecamere IP.

3.1.2.1 Protocollo base:

RTP funge da protocollo di base, basandosi a sua volta principalmente sul protocollo User Datagram Protocol (UDP), ma potendo essere implementato anche insieme al TCP.
Il numero di porta predefinito del livello di trasporto è 554 sia per TCP che per UDP.

3.1.2.2 Struttura:

Nel corso della nostra analisi di RTSP, utilizzeremo il termine per descrivere una serie di protocolli che lavorano insieme per la consegna di contenuti all'utente.

RTSP

RTSP è il protocollo di controllo per la consegna di contenuti multimediali attraverso le reti IP. Si basa tipicamente su TCP per una consegna affidabile e ha un funzionamento e una sintassi molto simili a HTTP. RTSP viene utilizzato dall'applicazione client per comunicare al server informazioni quali il file multimediale richiesto, il tipo di applicazione utilizzata dal client, il meccanismo di consegna del file (unicast o multicast, UDP o TCP) e altri importanti comandi informativi di controllo quali DESCRIBE, SETUP e PLAY. Il contenuto multimediale vero e proprio non viene in genere trasmesso attraverso le connessioni RTSP. RTSP è analogo al controllo remoto dei protocolli di streaming.

Protocollo di trasporto in tempo reale (RTP)

RTP è il protocollo utilizzato per il trasporto e la consegna dei dati audio e video in tempo reale. Poiché la consegna dei dati effettivi per l'audio e il video è tipicamente sensibile ai ritardi, come meccanismo di consegna viene utilizzato il protocollo UDP, più leggero, anche se in ambienti che soffrono di una maggiore perdita di pacchetti può essere utilizzato anche TCP. Il flusso RTP durante la consegna dei contenuti è unidirezionale dal server al client. Un aspetto interessante del funzionamento di RTP è che la porta di origine utilizzata dal server per l'invio dei dati UDP è sempre pari, sebbene sia assegnata dinamicamente. La porta di destinazione (cioè la porta UDP su cui il client è in ascolto) viene scelta dal client e comunicata tramite la connessione di controllo RTSP.

Protocollo di controllo in tempo reale (RTCP)

RTCP è un protocollo complementare a RTP ed è un meccanismo bidirezionale basato su UDP che consente al client di comunicare informazioni sulla qualità del flusso al server come: numero cumulativo di pacchetti persi, identificatori di inizio e fine del flusso, stima del jitter, gestione del throughput della sessione e latenza della trasmissione. La comunicazione UDP

RTCP utilizza sempre la porta sorgente UDP successiva a quella utilizzata dal flusso RTP e, di conseguenza, è sempre dispari.
La Figura successiva mostra il funzionamento congiunto dei tre protocolli.

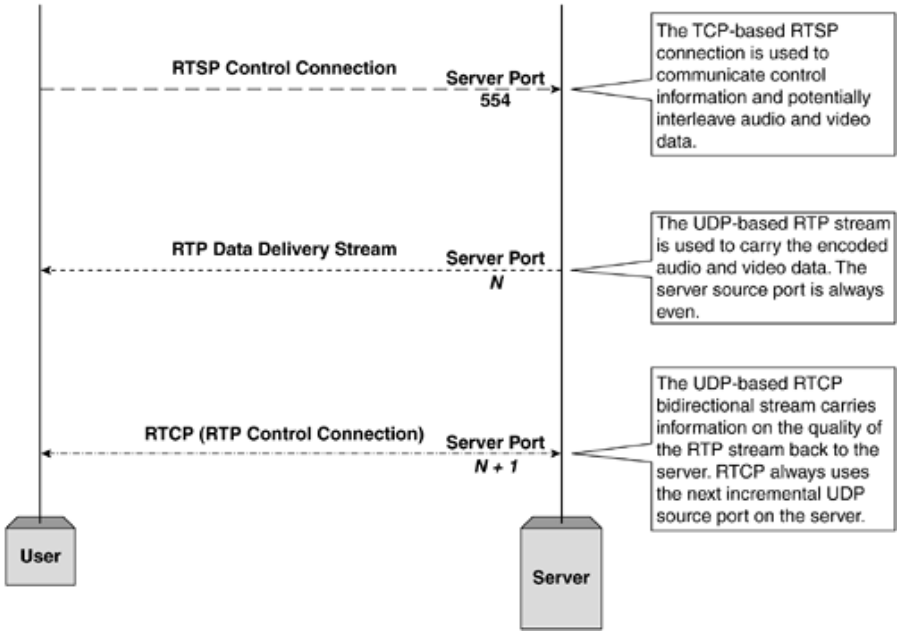


Figura 16 - I tre principali protocolli applicativi utilizzati nello streaming in tempo reale

3.1.2.3 Funzionamento:

Il protocollo RTSP è molto simile nella struttura e, in particolare, nella sintassi al HTTP. Entrambi utilizzano la stessa struttura di URL per descrivere un oggetto, con RTSP che utilizza lo schema rtsp:// anziché http://. RTSP, tuttavia, introduce una serie di header aggiuntivi (come DESCRIBE, SETUP e PLAY) e consente anche il trasporto dei dati out-of-band e su un protocollo diverso, come RTP descritto in precedenza.

Le fasi fondamentali del processo sono le seguenti:

1. Il client stabilisce una connessione TCP ai server, in genere sulla porta TCP 554, la porta nota (*well-know*) per RTSP.
2. Il client inizia quindi a emettere una serie di comandi di header RTSP che hanno un formato simile a quello di HTTP, ognuno dei quali viene riconosciuto dal server. All'interno di questi comandi RTSP, il client descrive al server i dettagli dei requisiti della sessione, come la versione di RTSP supportata, il trasporto da utilizzare per il flusso di dati e le informazioni sulle porte UDP o TCP associate. Queste informazioni vengono passate tramite le intestazioni DESCRIBE e SETUP e vengono completate

nella risposta del server con un ID di sessione che il client, ed eventuali dispositivi proxy transitori, possono utilizzare per identificare il flusso in ulteriori scambi.

3. Una volta completata la negoziazione dei parametri di trasporto, il client emette un comando PLAY per indicare al server di iniziare la consegna del flusso di dati RTP.
4. Una volta che il client decide di chiudere il flusso, viene emesso un comando TEARDOWN insieme all'ID di sessione che indica al server di interrompere la consegna di RTP associata a quell'ID.

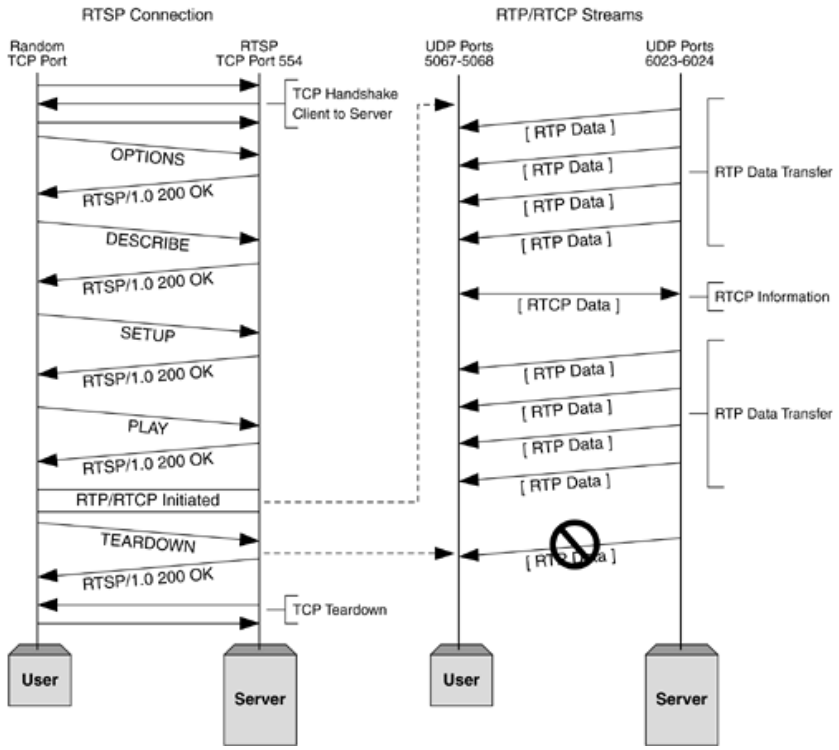


Figura 17 - Esempio di RTSP in azione, con i dati video/audio trasmessi su un flusso RTP UDP separato

3.2 Protocolli adattivi basati su HTTP:

I flussi video in live streaming distribuiti tramite HTTP non sono tecnicamente "flussi". Si tratta piuttosto di download progressivi inviati tramite normali server web. Questi protocolli possono gestire flussi di bitrate diverso utilizzando lo streaming a bitrate adattivo (ABR). Tra i protocolli basati su HTTP più comuni vi sono MPEG-DASH e HLS di Apple e le loro estensioni a bassa latenza, LL-HLS e LL-DASH.

3.2.1 HLS

HLS è il protocollo più utilizzato oggi per lo streaming. È stato originariamente rilasciato da Apple nel 2009 come parte dei suoi sforzi per eliminare Flash dagli iPhone. Questo protocollo è compatibile con un'ampia varietà di dispositivi, dai browser desktop, alle smart TV, ai set-top box, ai dispositivi mobili Android e iOS e persino ai lettori video HTML5.

HTTP Live Streaming (HLS) è un protocollo che sfrutta HTTP per distribuire file multimediali in piccole porzioni per ridurre il tempo di buffering e la latenza end-to-end. HLS risponde alle variazioni della velocità di trasferimento e regola, di conseguenza, la qualità dei media trasmessi grazie allo streaming a bitrate adattivo (ABR), consentendo allo streaming di continuare con una qualità video inferiore anche se la capacità di trasferimento diminuisce. Può aumentare la qualità una volta recuperata la capacità di trasferimento.

Viene utilizzato principalmente come *egress protocol* (in coppia con RTMP come *ingest protocol*) consentendo lo streaming di contenuti dal media server di elaborazione al riproduttore video del client.

3.2.1.1 Protocollo base:

HLS è basato sul protocollo Hypertext Transfer Protocol (HTTP) come protocollo di applicazione, che a sua volta presuppone un protocollo di livello di trasporto sottostante e affidabile, per cui viene comunemente utilizzato il Transmission Control Protocol (TCP). Tuttavia, l'HTTP può essere adattato per utilizzare protocolli non affidabili, come il protocollo UDP (User Datagram Protocol).

3.2.1.2 Struttura:

HLS è basato su file di playlist chiamati Media Playlist, ovvero file di testo contenenti URI (Uniform Resource Identifier). Una playlist multimediale contiene un elenco di segmenti multimediali (Media Segments) che, se riprodotti in sequenza, riproducono la presentazione multimediale.

Le Media Playlist sono contenute in genere in una Master Playlist che racchiude tutte le playlist che indicizzano lo stesso contenuto multimediale, con ogni playlist che adotta specifiche del contenuto (ad esempio risoluzione, bitrate etc.) adatte a client diversi.

Esempio di una Media Playlist:

```
#EXTM3U
#EXT-X-TARGETDURATION:10

#EXTINF:9.009,
http://media.example.com/first.ts
#EXTINF:9.009,
http://media.example.com/second.ts
#EXTINF:3.003,
http://media.example.com/third.ts
```

Figura 18 - La prima riga è il tag identificativo di formato #EXTM3U. La riga contenente #EXT-X-TARGETDURATION indica che tutti i segmenti multimediali avranno una durata massima di 10 secondi. Quindi, vengono dichiarati tre segmenti multimediali. Il primo e il secondo durano 9,009 secondi; il terzo 3,003 secondi.

Una playlist multimediale contiene una serie di segmenti multimediali che costituiscono la presentazione complessiva. Un segmento multimediale è specificato da un URI e, facoltativamente, da un intervallo di byte, mentre la durata di ogni segmento multimediale è indicata nella playlist multimediale dal suo tag EXTINF.

Ogni segmento multimediale di una playlist multimediale ha un identificativo univoco e sequenziale. Il Media Sequence Number del primo segmento della playlist multimediale è 0 o è dichiarato nella playlist.

Ogni segmento multimediale contiene la continuazione del flusso di bit codificati dalla fine del segmento con il numero di sequenza multimediale precedente. Le uniche eccezioni sono il primo segmento multimediale che appare in una playlist multimediale e i segmenti multimediali segnalati esplicitamente come discontinuità.

La durata di una playlist multimediale è la somma delle durate dei segmenti multimediali al suo interno. La velocità in bit di un segmento multimediale è la dimensione del segmento multimediale divisa per la sua durata EXTINF. Questo valore include l'overhead del contenitore, ma non include l'overhead imposto dal sistema di consegna, come le intestazioni HTTP, TCP o IP.

La velocità in bit di picco di un segmento di una playlist multimediale è la velocità in bit maggiore di qualsiasi insieme contiguo di segmenti la cui durata totale è compresa tra 0,5

volte la durata target e 1,5 volte la durata target più 0,5 secondi (poiché i segmenti multimediali possono superare la durata target fino a 0,5 secondi). La velocità in bit di un set è calcolata dividendo la somma delle dimensioni dei segmenti per la somma delle durate dei segmenti.

Il bit rate medio dei segmenti di una playlist multimediale è la somma delle dimensioni (in bit) di ogni segmento multimediale della playlist, diviso per la durata della playlist.

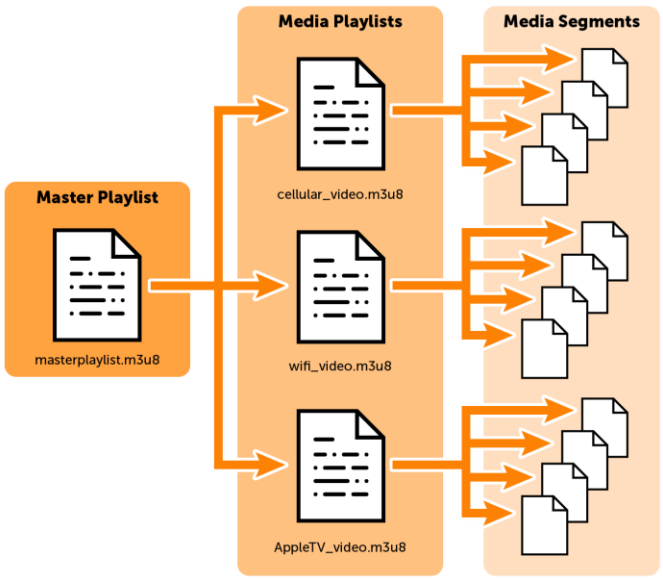


Figura 19 - Suddivisione di una Master Playlist in Media Playlist e Media Segments

3.2.1.3 Funzionamento:

1. Il client richiede al server di avviare la riproduzione del contenuto
2. Il server risponde fornendo il file di indice, la Media Playlist, contenente gli URI dei segmenti video sequenziali.
3. Il client, leggendo gli indirizzi nell'indice, avvia il download sequenziale dei file video. Il primo segmento caricato è generalmente quello che il client ha scelto di riprodurre per primo. Per riprodurre normalmente la presentazione, il segmento multimediale successivo da caricare è quello con il numero di sequenza multimediale più basso che è maggiore del numero di sequenza multimediale dell'ultimo segmento multimediale caricato.
4. Il player video inizia la riproduzione dei segmenti video in ordine.

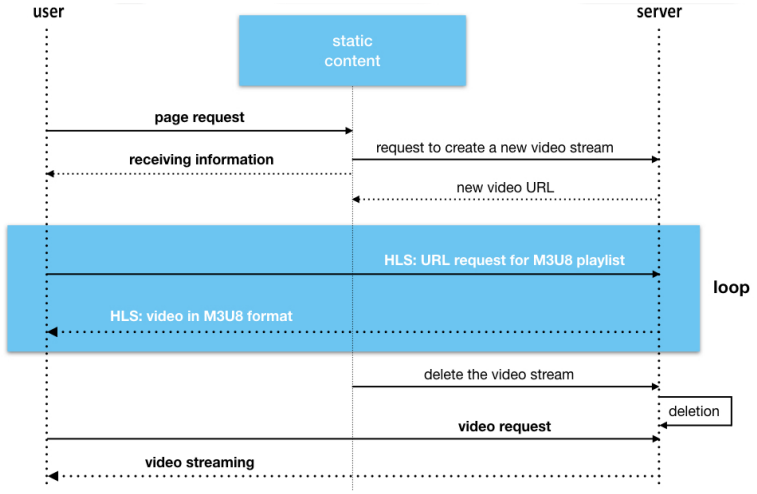


Figura 20 - Schema dei messaggi scambiati in un flusso con protocollo HLS

3.2.2 LL-HLS

Estensione a bassa latenza del protocollo HLS, il protocollo Low Latency HLS fornisce un canale parallelo per la distribuzione dei media nella playlist multimediale che permette di abbattere la metrica della latenza vista nel capitolo 2.

La riproduzione live con un ritardo ridotto richiede alcune caratteristiche del flusso e del trasporto per supportare la consegna tempestiva dei media. I client devono verificare che il server soddisfi questi requisiti prima di riprodurre con un ritardo inferiore a due durate target (la durata specificata per un segmento). Poiché le estensioni a bassa latenza sono aggiunte piuttosto che sostituzioni, i client possono tornare alla riproduzione a latenza normale se scoprono che il server non soddisfa i requisiti di questo profilo di configurazione.
HLS a bassa latenza divide i media in un numero maggiore di file più piccoli, come i CMAF Chunks (il Common Media Application Format o CMAF è un formato per pacchettizzare e distribuire varie forme di media basati su HTTP) o i file .TS. Questi file più piccoli sono chiamati segmenti parziali. Poiché ogni segmento parziale ha una durata breve, può essere confezionato, pubblicato e aggiunto alla playlist multimediale molto prima del suo segmento "padre".

Mentre i normali segmenti multimediali possono durare 6 secondi ciascuno, un esempio di segmento parziale può essere di soli 200 millisecondi. Un primo segmento parziale potrebbe essere pubblicato solo 200 millisecondi dopo la pubblicazione del segmento precedente, seguito da 29 suoi simili e infine da un segmento multimediale di lunghezza regolare, di 6 secondi, contenente lo stesso media della concatenazione dei suoi 30 segmenti parziali. Per

ridurre l'ingombro della playlist, il server rimuove i segmenti parziali dalla playlist multimediale una volta che sono superiori (più vecchi) a tre durate target.

I segmenti parziali vengono aggiunti alla playlist multimediale utilizzando il nuovo tag EXT-X-PART.

Le principali differenze con il normale HLS sono:

- Ogni segmento multimediale può essere suddiviso a sua volta in segmenti “figli” chiamati segmenti parziali. Poiché ogni segmento parziale ha una durata breve, può essere pacchettizzato, pubblicato e aggiunto alla playlist multimediale molto prima del suo segmento “padre”.
- I flussi live comportano frequenti download di playlist. Quando le playlist sono di grandi dimensioni e un client possiede già la versione precedente, il costo di trasferimento può essere ridotto in modo considerevole inviando solo le informazioni più recenti in risposta a una richiesta di aggiornamento della playlist. Invece della playlist completa, viene inviata la differenza tra le playlist (nota anche come delta) che contiene solo i nuovi segmenti.
- La scoperta della disponibilità di nuovi segmenti per un live stream HLS viene solitamente effettuata dal client ricaricando il file della playlist a intervalli regolari e controllando l'aggiunta di nuovi segmenti. Nel caso di streaming a bassa latenza, è auspicabile evitare qualsiasi ritardo tra la disponibilità di un segmento (parziale) nella playlist e la scoperta della sua disponibilità da parte del client. Con l'approccio di ricarica della playlist, tale ritardo può essere pari all'intervallo di tempo di ricarica nel caso peggiore. Con la nuova funzione di blocco delle ricariche delle playlist, i client possono specificare la disponibilità di un segmento futuro che stanno aspettando e il server dovrà trattenere la richiesta di playlist finché quel segmento specifico non sarà disponibile nella playlist. Il segmento da attendere viene specificato con un parametro di query nella richiesta di playlist.
- I segmenti parziali presto disponibili sono pubblicizzati prima della loro effettiva disponibilità nella playlist da un nuovo tag EXT-X-PRELOAD-HINT. Ciò consente ai client di aprire una richiesta in anticipo e il server risponderà non appena i dati saranno disponibili. In questo modo il client può "risparmiare" il tempo di andata e ritorno della richiesta.

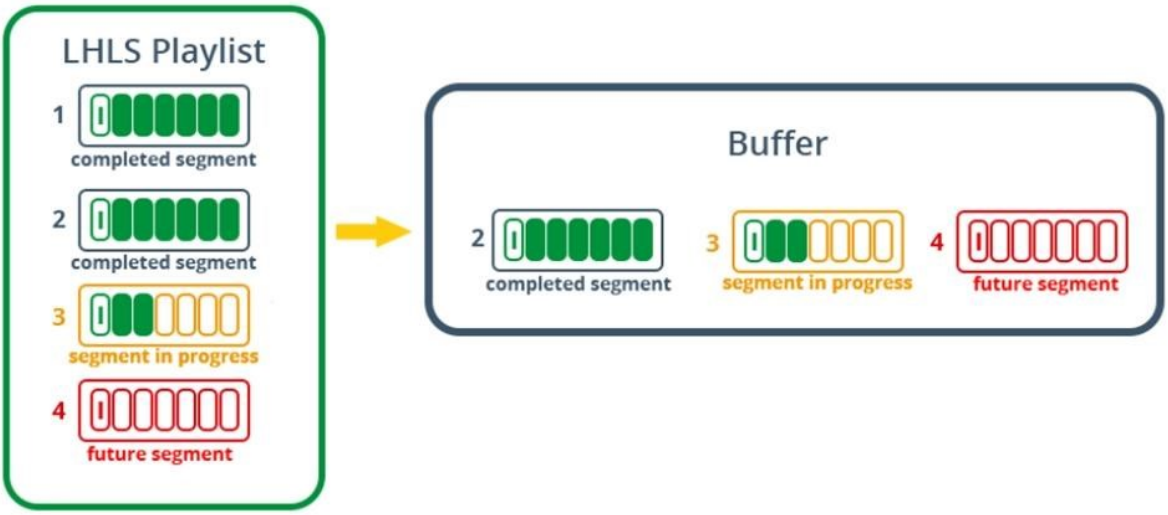


Figura 21 - Schema di riempimento di un buffer con LL-HLS

3.2.3 DASH

Dynamic adaptive streaming over HTTP (DASH), conosciuto anche come MPEG-DASH, rappresenta un'alternativa indipendente e non proprietaria a HLS pur utilizzando una struttura e un funzionamento simile. DASH è la prima soluzione di streaming adattivo basata su HTTP a diventare uno standard internazionale.

3.2.3.1 Protocollo base:

DASH è basato sul protocollo di applicazione HTTP, mentre il protocollo di trasporto utilizzato è TCP. Tuttavia, come nel caso del protocollo HLS, HTTP può essere adattato per utilizzare protocolli non affidabili, come il protocollo UDP (User Datagram Protocol).

3.2.3.2 Struttura:

DASH si basa su file di indice chiamati Media Presentation Description (MPD). Il MPD fornisce informazioni sufficienti affinché un client possa fornire un servizio di streaming all'utente accedendo ai segmenti in esso indirizzati attraverso il protocollo HTTP/1.1. Un client di questo tipo viene definito client DASH.

L'aspetto particolare di "HTTP" in DASH è l'uso di URL HTTP nel MPD per fare riferimento ai segmenti multimediali. L'uso di HTTP-URL consente di ottenere informazioni univoche sulla posizione e fornisce metodi ben definiti per accedere alle risorse, in particolare HTTP GET e HTTP partial GET.

Ogni MPD descrive una Media Presentation, ovvero una presentazione multimediale. Il contenuto multimediale è composto da uno o più *periods* (periodi) di contenuto multimediale contigui nel tempo. Il contenuto di diversi periodi di contenuto multimediale può essere completamente indipendente o alcuni periodi di una presentazione multimediale possono appartenere allo stesso Asset.
Ogni periodo di contenuto multimediale è composto da uno o più componenti di contenuto multimediale, ad esempio componenti audio in varie lingue, diversi componenti video che forniscono diverse visioni dello stesso programma, sottotitoli in varie lingue, ecc. A ogni componente di contenuto multimediale è assegnato un tipo di componente di contenuto multimediale, ad esempio audio o video. Ogni componente di un contenuto multimediale può avere diverse versioni codificate, denominate flussi multimediali.

Il MPD descrive la sequenza di periodi nel tempo che compongono la presentazione multimediale. Un periodo rappresenta tipicamente un periodo di contenuto multimediale durante il quale è disponibile un insieme coerente di versioni codificate del contenuto multimediale, ossia l'insieme di bitrate, lingue, didascalie, sottotitoli ecc. disponibili non cambia durante un periodo.

All'interno di un periodo, il materiale è organizzato in set di adattamento (Adaptation Sets). Un set di adattamento rappresenta un insieme di versioni codificate intercambiabili di uno o più componenti del contenuto multimediale. Ad esempio, può esistere un set di adattamento per il componente video principale e uno separato per il componente audio principale. Se sono disponibili altri materiali, ad esempio didascalie o descrizioni audio, questi possono avere un set di adattamento separato. Il materiale può anche essere fornito in forma multipla, nel qual caso le versioni intercambiabili del multiplex possono essere descritte come un unico set di adattamento. Ciascuna delle componenti del multiplex può essere descritta individualmente da una descrizione del contenuto multimediale.

Un set di adattamento contiene un insieme di rappresentazioni. Una rappresentazione descrive una versione codificata adatta alla consegna di uno o più componenti di contenuto multimediale. Una rappresentazione include uno o più flussi multimediali. Ogni singola rappresentazione all'interno di un Adaptation Set è sufficiente per eseguire il rendering dei componenti di contenuto multimediale al suo interno. Raccogliendo diverse rappresentazioni in un Adaptation Set, la sorgente della presentazione multimediale indica che le rappresentazioni sono contenuti percettivamente equivalenti. In genere, ciò significa che i

client possono passare dinamicamente da una rappresentazione all'altra all'interno di un Adaptation Set per adattarsi alle condizioni della rete o ad altri fattori. Il passaggio si riferisce alla presentazione di dati decodificati fino a un certo tempo t e alla presentazione di dati decodificati di un'altra rappresentazione dal tempo t in poi. Se le rappresentazioni sono incluse in un Adaptation Set e il client commuta correttamente, ci si aspetta che la presentazione multimediale sia percepita senza soluzione di continuità attraverso lo switch. I client DASH possono ignorare le rappresentazioni che contengono codec o tecnologie di rendering non supportate o che sono altrimenti inadatte.

All'interno di una rappresentazione, il contenuto può essere suddiviso nel tempo in segmenti per una corretta accessibilità e consegna. Per accedere a un segmento, viene fornito un URL per ogni segmento. Di conseguenza, un segmento è la più grande unità di dati che può essere recuperata con una singola richiesta HTTP. Per le rappresentazioni segmentate, si distinguono due tipi di segmenti: i segmenti di inizializzazione contengono metadati statici per la rappresentazione, mentre i segmenti multimediali contengono campioni multimediali e avanzano nella timeline. Le rappresentazioni possono anche essere organizzate da un singolo segmento auto-inizializzante che contiene sia le informazioni di inizializzazione che i dati multimediali.

Ai segmenti viene assegnata una durata, che è la durata del media contenuto nel segmento quando viene presentato a velocità normale. In genere, tutti i segmenti di una rappresentazione hanno una durata uguale o più o meno simile. Tuttavia, la durata dei segmenti può variare da una rappresentazione all'altra. Una presentazione DASH può essere costruita con segmenti relativamente brevi (ad esempio pochi secondi), oppure con segmenti più lunghi che includono un singolo segmento per l'intera rappresentazione.
I segmenti brevi sono solitamente richiesti nel caso di contenuti dal vivo, dove ci sono restrizioni sulla latenza end-to-end. La durata di un segmento è tipicamente un limite inferiore alla latenza end-to-end. DASH non supporta la possibilità di estendere i segmenti nel tempo: un segmento è trattato come un oggetto, come un'unità completa e discreta che viene resa disponibile nella sua interezza.

I client possono passare da una rappresentazione all'altra all'interno di un set di adattamento in qualsiasi punto del supporto. Tuttavia, il passaggio in posizioni arbitrarie può essere complesso a causa delle dipendenze di codifica all'interno delle rappresentazioni e di altri fattori, richiedendo potenzialmente il download e la decodifica paralleli nel client DASH.

La segmentazione e la sottosegmentazione possono essere eseguite in modo da semplificare la commutazione. Ad esempio, nei casi più semplici, ogni segmento o sottosegmento inizia con uno stream access point (SAP) e i confini dei segmenti o sottosegmenti sono allineati tra le rappresentazioni di un set di adattamento. In questo caso, il cambio di rappresentazione comporta la riproduzione fino alla fine di un (sotto)segmento di una rappresentazione e quindi la riproduzione dall'inizio del (sotto)segmento successivo della nuova rappresentazione. La descrizione della presentazione multimediale e l'indice dei segmenti forniscono varie indicazioni che descrivono proprietà delle rappresentazioni che possono semplificare la commutazione.

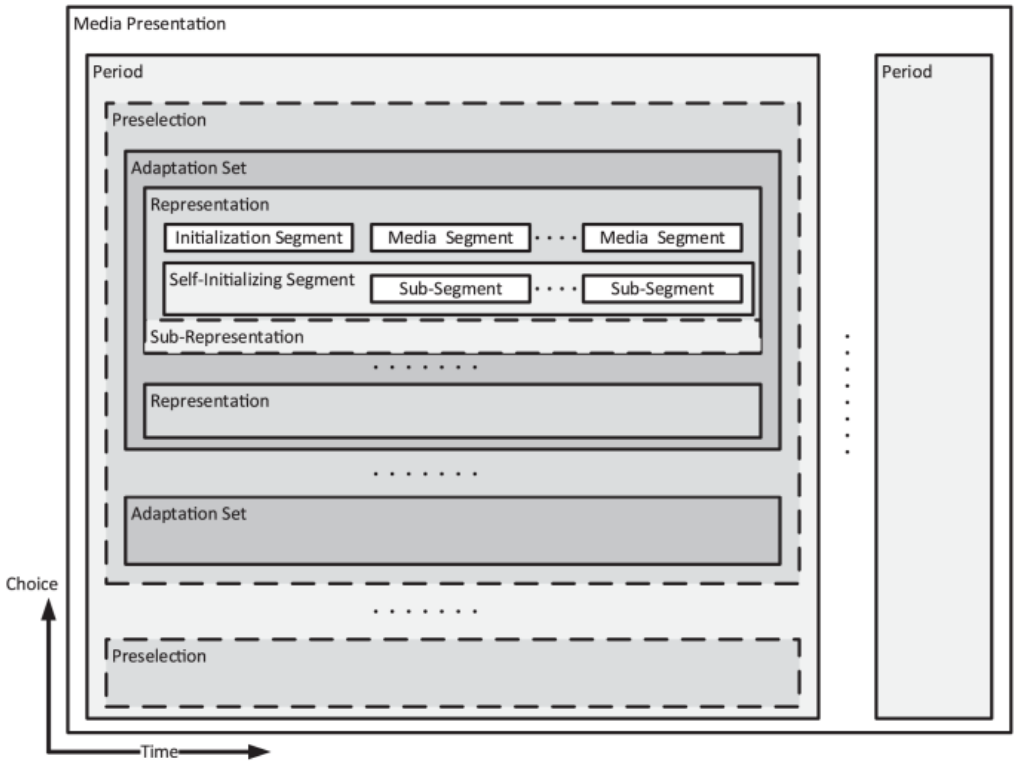


Figura 22 - Nella figura è rappresentata la gerarchia di contenitori descritta.

3.2.3.3 Funzionamento:

Come suggerisce il nome, il DASH funziona secondo i principi dell'ABR. Utilizzando le metriche descritte nel capitolo 2 attraverso l'implementazione di un algoritmo di adattamento scelto dal client, lo streaming attraverso DASH può ottimizzare la ricezione del flusso dati al client.

- La sorgente genera e codifica un contenuto che viene successivamente confezionato da un servizio o software di packaging DASH. Il packager divide ogni pacchetto in piccoli segmenti di una durata specifica (ad esempio, 2 secondi o 4 secondi).

- Il packager registra anche il modo in cui ha suddiviso i video e l'ordine in cui devono essere consegnati nel MPD.
- Il contenuto e il MPD sono memorizzati su un server di origine in attesa di essere consegnati a un lettore, di solito tramite una content delivery network (CDN).
- All'altro capo, c'è un lettore conforme a DASH con un motore di streaming ABR incorporato.
- Quando l'utente attiva la riproduzione, l'applicazione o il lettore video richiede il file MPD del video. Dopo aver ricevuto il file MPD, il lettore lo analizza per stabilire come riprodurre il video.
- Il lettore inizia quindi a richiedere i segmenti del video nell'ordine predefinito, li decodifica e visualizza il video all'utente.
- Inoltre, il player tramite un algoritmo di adattamento (solitamente implementato come una misura della velocità in bit calcolata utilizzando il peso del segmento scaricato diviso il tempo impiegato per il download) monitora continuamente le condizioni del throughput al client. In base alla banda disponibile, il lettore sceglie una delle velocità di trasmissione pubblicizzate nel MPD e chiede alla CDN di inviare il successivo segmento di video da quella variante.

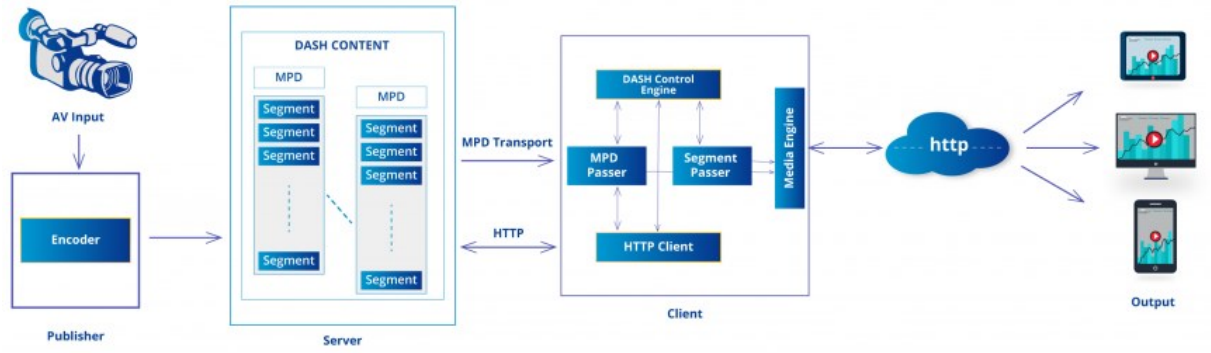


Figura 23 - Schema di trasmissione completo per uno streaming con protocollo DASH

3.2.4 LL-DASH

Il protocollo Low Latency DASH costituisce un'estensione del normale DASH in cui i segmenti vengono tagliati in pezzi più piccoli e non sovrapposti (spesso contenenti una manciata di fotogrammi), tutti indipendenti l'uno dall'altro. Ciò significa che l'origine non deve aspettare che il segmento sia completamente finito prima di poter inviare i primi pezzi al client.

Il MPD può indicare quando un segmento sarà disponibile sul server. Osservando quando i segmenti diventano disponibili, un client può richiedere rapidamente e accuratamente un segmento non appena diventa disponibile, riducendo i ritardi. Ciò richiede una sincronizzazione accurata tra gli orologi del client e del server. A tale scopo, MPEG-DASH consente di configurare un *time server*.

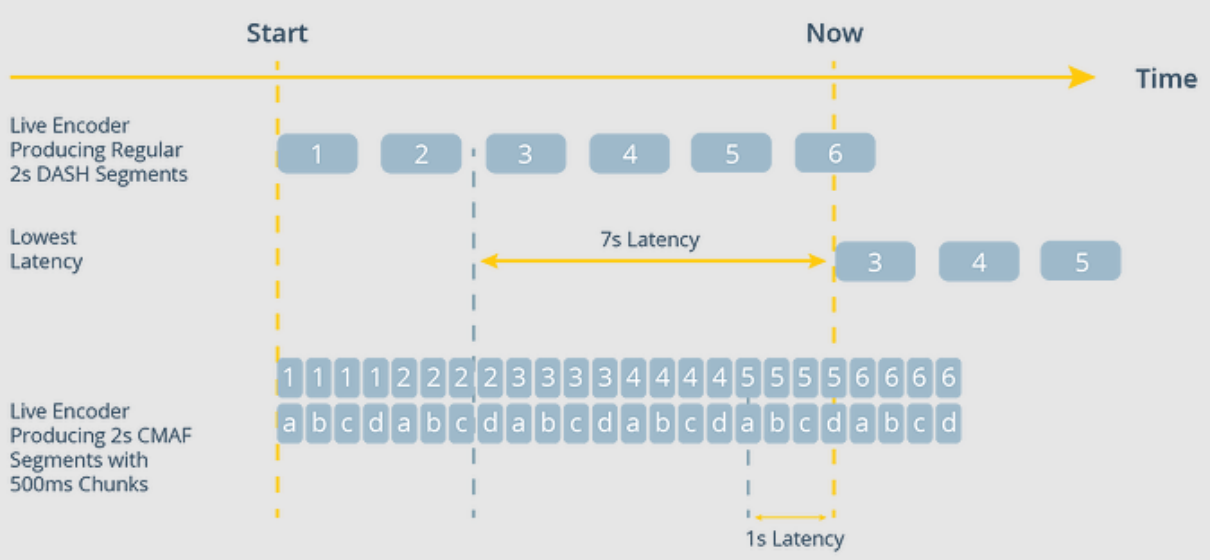


Figura 24 - Differenza nella durata dei segmenti tra DASH e LL-DASH

Di solito, durante lo streaming live, i media vengono inviati in segmenti di pochi secondi ciascuno. Questi segmenti, secondo le specifiche del protocollo DASH, non dovrebbero essere più corti di 2 o 4 secondi, altrimenti si rischia di avere prestazioni e qualità di riproduzione scadenti. Con LL-DASH, se il lettore richiede un segmento che non è stato completato, i segmenti saranno consegnati non appena disponibili utilizzando la codifica Chunked Transfer, ovvero un meccanismo di trasferimento dati codificato in HTTP. Poiché i media possono essere consegnati al client non appena vengono generati lato server, l'overhead viene ridotto, consentendo di ridurre il buffer del client e, di conseguenza, la latenza. Il buffer di un client, secondo le specifiche del protocollo, dovrebbe contenere circa 3-4 segmenti per garantire una riproduzione fluida, riducendo potenzialmente la latenza sullo stesso ordine di grandezza. In pratica, è importante tenere conto anche della gestione degli MPD, della sincronizzazione temporale e del buffering per i problemi di rete. Tramite LL-DASH, la latenza può essere ridotta fino a un paio di secondi invece dei 10 secondi che sono ottenibili mediamente con il solo DASH.

3.3 Protocolli di ultima generazione basati su versioni evolute di TCP/UDP:

3.3.1 SRT

SRT è un protocollo open-source sviluppato da Haivision, un fornitore di tecnologia streaming, e basato su un'evoluzione di UDT, un protocollo di Automatic Repeat reQuest (ARQ). Il protocollo è già utilizzato dai membri della SRT Alliance, un insieme sempre più crescente di aziende esperte del mercato digitale. Questo protocollo non si basa su un singolo codec e consente agli sviluppatori di abbinarlo a qualsiasi codec audio e video.

SRT collega due endpoint allo scopo di fornire video e altri flussi multimediali a bassa latenza attraverso reti IP con perdita di dati, come la rete Internet pubblica.

3.3.1.1 Protocollo base:

Come accennato, SRT è un protocollo derivato da UDT: fornisce connessione, controllo e trasmissione affidabile simile a TCP; tuttavia, lo fa a livello di applicazione, utilizzando il protocollo UDP come livello di trasporto sottostante. SRT supporta il recupero dei pacchetti mantenendo una bassa latenza (default: 120 ms).

3.3.1.2 Struttura:

SRT utilizza pacchetti strutturati secondo il protocollo UDP, ma con alcune modifiche. Ogni pacchetto UDP che trasporta il traffico SRT contiene un header SRT (immediatamente successivo a quello UDP).
SRT ha due tipi di pacchetti, i pacchetti di dati e i pacchetti di controllo.

I pacchetti vengono inseriti in un indice non appena sono inseriti nel buffer di invio. Ogni pacchetto ha il suo timestamp locale che viene utilizzato per stabilire quando deve essere inviato.
A certi intervalli il destinatario invia degli ACK che vengono utilizzati per rimuovere i pacchetti già correttamente ricevuti dal buffer del mittente.

Il timestamp nel header del pacchetto UDP viene utilizzato dal ricevitore per stabilire se il pacchetto ricevuto è ancora valido o se è un pacchetto in ritardo rispetto al flusso già riprodotto e quindi scartato immediatamente.

Negli ambienti di rete fortemente lossy SRT utilizza dei Periodic NAK Reports, inviati con un periodo di RTT/2 e un floor minimo di 20 ms, per mitigare gli effetti della perdita di pacchetti. È possibile recuperare fino al 10% di perdita di pacchetti entro i limiti dei parametri di configurazione SRT.
Un pacchetto di controllo NAK contiene un elenco compresso dei pacchetti persi. Pertanto, solo i pacchetti persi vengono ritrasmessi. Utilizzando RTT/2 per il periodo di segnalazione dei NAK, può accadere che i pacchetti persi vengano ritrasmessi più di una volta, ma ciò contribuisce a mantenere una bassa latenza nel caso in cui i pacchetti NAK vengano persi.

La peculiarità di SRT è quindi il suo approccio orientato al recupero dei pacchetti persi tramite lo sfruttamento di varie feature codificate a partire dal protocollo UDT4 [15] che permettono una gestione aggiuntiva dei pacchetti persi o ritardati rispetto all'utilizzo del solo UDP.

3.3.1.3 Funzionamento:

La sorgente produce il contenuto multimediale che viene passato all'encoder. Qui il codificatore utilizzando il protocollo SRT spezza il contenuto multimediale in piccoli pacchetti, ognuno con un header SRT. I pacchetti dati vengono spediti al ricevitore, che può essere un server nel caso si utilizzi il protocollo come *ingest protocol* o un client nel caso di *egress protocol.* Al ricevente vengono gestiti sfruttando le informazioni del header SRT.
Il diagramma seguente fornisce una panoramica di alto livello dello scambio di dati (compresi i dati di controllo) tra due peer in una sessione SRT punto-punto. Si noti che i ruoli dei peer cambiano nel corso della sessione. Ad esempio, i peer possono iniziare come Caller e Listener durante l'handshake, ma poi diventare Sender e Receiver per la parte di trasmissione dei dati.

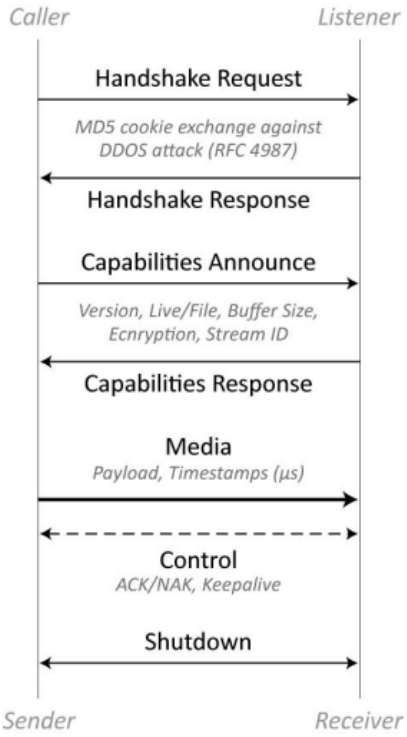


Figura 25 - Scambio dati con SRT

3.3.2 WebRTC

Web Real-Time Communication (WebRTC) è una tecnologia open source creata da Google e standardizzata nel gennaio 2021, spesso utilizzata per le videoconferenze. Il protocollo WebRTC incorpora tecnologie di comunicazione in tempo reale all'interno dei browser web utilizzando JavaScript, API e Hypertext Markup Language (HTML). Per la prima volta, i browser sono in grado di scambiare direttamente contenuti multimediali in tempo reale con altri browser in modalità peer-to-peer (P2P).

In genere, WebRTC connette gli utenti trasferendo flussi audio, video e dati in tempo reale da un dispositivo all'altro utilizzando il P2P. Tuttavia, se gli utenti si trovano su reti Internet Protocol (IP) diverse con firewall NAT (Network Address Translation) che impediscono la comunicazione in tempo reale (RTC), è possibile utilizzare i server STUN (Session Traversal Utilities for NAT) per tradurre un indirizzo IP in un indirizzo Internet pubblico, in modo da stabilire connessioni P2P. Le API WebRTC avviano e monitorano le connessioni P2P tra dispositivi tramite browser e facilitano il trasferimento bidirezionale dei dati su più canali.

3.3.2.1 Protocollo base:

In WebRTC il trasporto del contenuto multimediale si basa sul protocollo RTP, in particolare il suo profilo utilizzato per la crittografia Secure Real-time Transport Protocol (SRTP), insieme a RTCP per la consegna dei flussi multimediali.

3.3.2.2 Struttura:

In WebRTC entrambi i browser eseguono un'applicazione web, scaricata da un server web diverso. I messaggi di segnalazione sono utilizzati per impostare e terminare le comunicazioni. Sono trasportati dal protocollo HTTP o WebSocket attraverso server web che possono modificarli, tradurli o gestirli a seconda delle necessità. Vale la pena notare che la segnalazione tra browser e server non è standardizzata in WebRTC, poiché è considerata parte dell'applicazione. Per quanto riguarda il percorso dei dati, una PeerConnection consente ai media di fluire direttamente tra i browser senza l'intervento di alcun server. I due server Web possono comunicare utilizzando un protocollo di segnalazione standard come SIP o Jingle (XEP-0166). Altrimenti, possono utilizzare un protocollo di segnalazione proprietario.

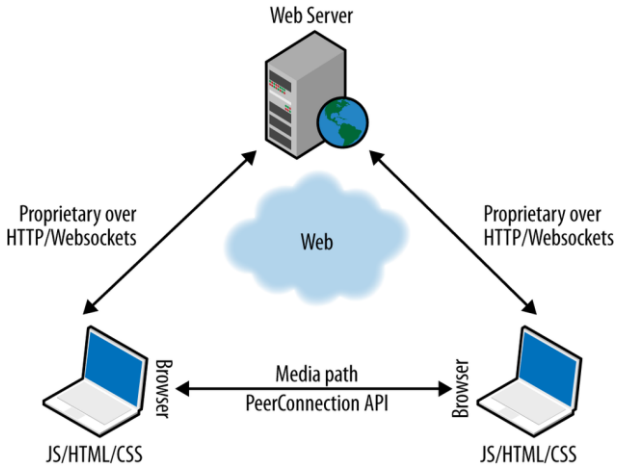


Figura 26 - Schematizzazione di un collegamento con WebRTC

La cattura del contenuto multimediale attraverso il browser viene effettuata da un set di API JavaScript.

L'interfaccia MediaStream è utilizzata per rappresentare flussi di dati multimediali. I flussi possono essere di ingresso o di uscita, nonché locali o remoti (ad esempio, una webcam locale o una connessione remota). Va notato che un singolo MediaStream può contenere zero o più tracce. Ogni traccia ha un oggetto MediaStreamTrack corrispondente che rappresenta una specifica sorgente multimediale nell'interprete. Tutte le tracce di un MediaStream devono essere sincronizzate durante il rendering. Un MediaStreamTrack rappresenta un contenuto che comprende uno o più canali, dove i canali hanno una relazione definita e nota tra loro. Un canale è l'unità più piccola considerata in questa specifica API.

La figura seguente mostra un MediaStream composto da una singola traccia video e due tracce audio distinte (canale destro e sinistro).

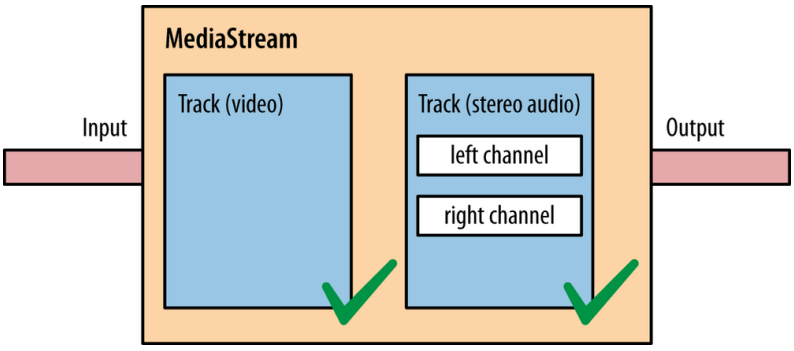


Figura 27 - Struttura di MediaStream

3.3.2.3 Funzionamento:

Gli utenti che desiderano stabilire un flusso live streaming tra loro devono connettersi allo stesso servizio che implementa WebRTC. A questo punto si stabilirà una connessione nel modo seguente:

1. Il browser dei due peer si connette alla pagina web del servizio chiamante e scarica una pagina HTML contenente un JavaScript che mantiene il browser connesso al server tramite una connessione sicura HTTP o WebSocket.
2. Quando la chiamata tra utenti viene avviata il codice JavaScript istanzia un oggetto PeerConnection. Una volta creata la PeerConnection, il codice JavaScript sul lato del servizio chiamante deve impostare i contenuti multimediali e svolge questo compito attraverso la funzione MediaStream. I contenuti iniziano ad essere raccolti (ad esempio tramite webcam e microfono).
3. Una volta creati alcuni flussi, il browser genera un messaggio di segnalazione che viene inviato al server di segnalazione (ad esempio, tramite XMLHttpRequest o WebSocket).
4. Il server di segnalazione elabora il messaggio dal browser, determina che si tratta di una chiamata al secondo utente e invia un messaggio di segnalazione al browser del destinatario.
5. Il JavaScript del browser destinatario elabora il messaggio in arrivo e avvisa l'utente. Se l'utente decide di rispondere alla chiamata, il JavaScript in esecuzione nel suo browser istanzia una PeerConnection relativa al messaggio proveniente dal primo utente. A questo punto si verifica un processo simile a quello del browser del primo utente. Il browser destinatario verifica che il servizio chiamante sia approvato e che i flussi multimediali siano creati; successivamente, un messaggio di segnalazione contenente informazioni sui media, i candidati Interactive Connectivity Establishment (ICE) e un'impronta digitale viene inviato al primo utente chiamante tramite il servizio di segnalazione.

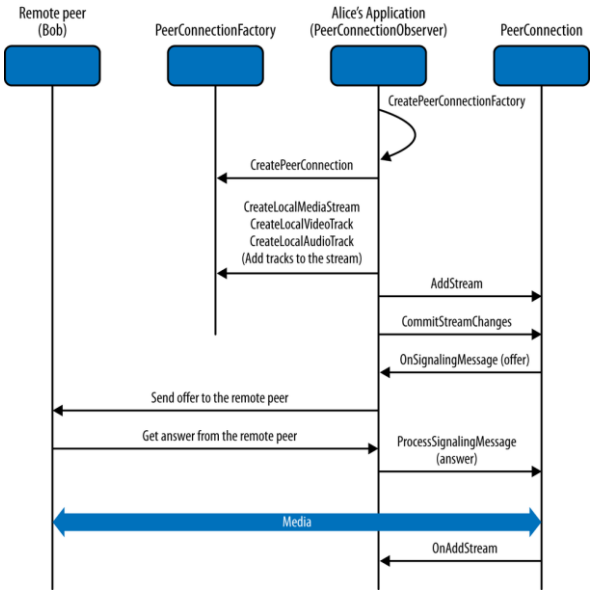


Figura 28 - Scambio dati in una connessione con WebRTC

3.3.3 RIST

Reliable Internet Stream Transport (RIST) è un protocollo di trasporto open-source e con specifiche aperte, progettato per la trasmissione affidabile di video e audio su reti lossy (incluso Internet) con bassa latenza e alta qualità.

Rilasciato nel 2018, RIST si basa in modo nativo sul protocolli ben noti come RTP e SMPTE-2022 (Transport Stream over IP), nonché su una serie di RFC (standard internet) rilevanti. Tutt'ora rimane ancora in aperto sviluppo.

In quanto tale, RIST può essere inteso come un insieme di linee guida basate sugli standard, messe a punto per l'applicazione in un ambiente broadcast. Questo approccio aperto significa che i fornitori possono ancora essere innovativi e differenziare le loro soluzioni quando implementano RIST, pur sapendo che funzionerà con altre soluzioni, indipendentemente dalla tecnologia utilizzata.

3.3.3.1 Protocollo base:

RIST fornisce un trasporto multimediale affidabile e ad alte prestazioni utilizzando RTP / UDP al livello di trasporto per evitare le limitazioni del TCP. L'affidabilità è ottenuta utilizzando ritrasmissioni basate su NACK (ARQ).

3.3.3.2 Struttura:

Un sistema RIST di base consiste in un mittente (A) e in un ricevitore (B) collegati attraverso una rete potenzialmente soggetta a perdite, eventualmente con percorsi di trasmissione multipli abilitati. RIST utilizza un protocollo di ritrasmissione selettiva basato sul NACK per recuperare la perdita di pacchetti.

3.3.3.3 Funzionamento:

Il funzionamento generale di questo protocollo è il seguente:

- I ricevitori non comunicano con i mittenti a meno che non rilevino una perdita di pacchetti.
- Una volta rilevata la perdita di un pacchetto, i ricevitori richiedono la ritrasmissione del pacchetto o dei pacchetti persi.
- I ricevitori implementano un buffer per gestire uno o più ritardi di andata e ritorno della rete e il riordino dei pacchetti.

- I pacchetti possono essere richiesti più volte, a condizione che il buffer del decodificatore sia sufficientemente ampio da permettere al
- pacchetti recuperati nella sequenza corretta nel flusso di decodifica.

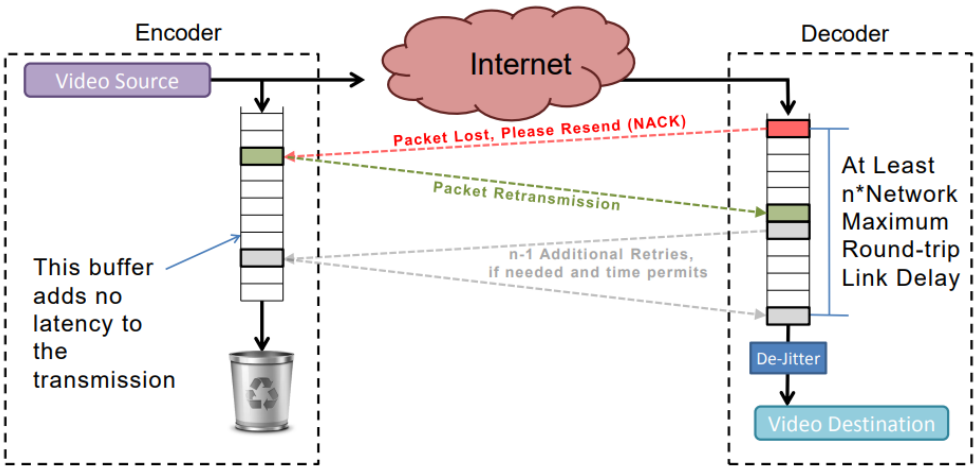


Figura 29 - Scambio pacchetti in una connessione RIST

3.4 Confronto tra protocolli:

3.4.1 RTMP, RTSP e HLS

Uno studio [22] ha confrontato tra loro i protocolli RTMP, RTSP e HLS in un contesto di utilizzo IPTV in reti mobili.

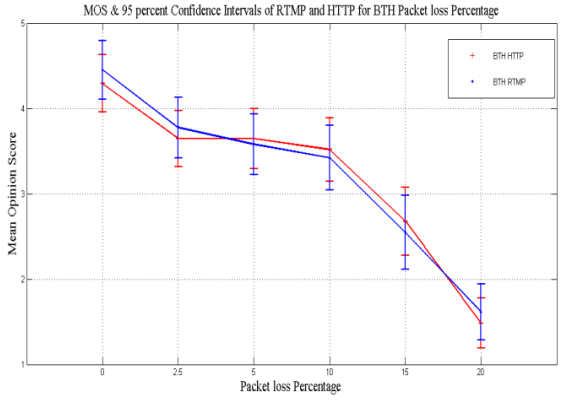
In particolare, è stato utilizzato un video in 720p a 30 FPS ad un bitrate di 8000kbps con codifica H.264, e per HLS sono state fornite anche le qualità inferiori di 240p, 360p e 480p ad un bitrate rispettivo di 5600kbps, 3400kbps e 200kbps. In tutti i casi gli autori hanno compiuto un viaggio in auto per attraversare le diverse coperture 2G, 3G e 4G e osservare il comportamento.

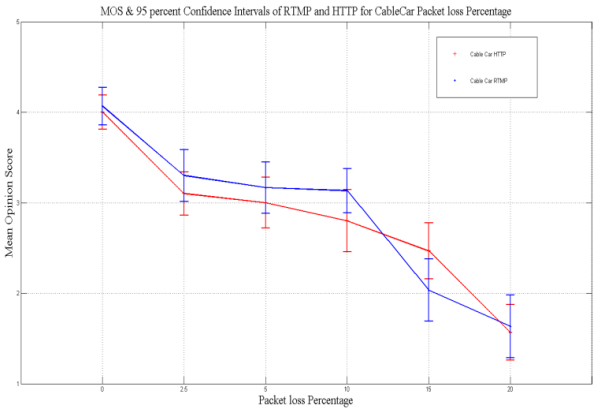
I risultati hanno dimostrato come HLS sia il protocollo più adatto in termini di stabilità in questo tipo di contesto essendo stato l'unico a non subire blocchi grazie alla sua natura adattiva.

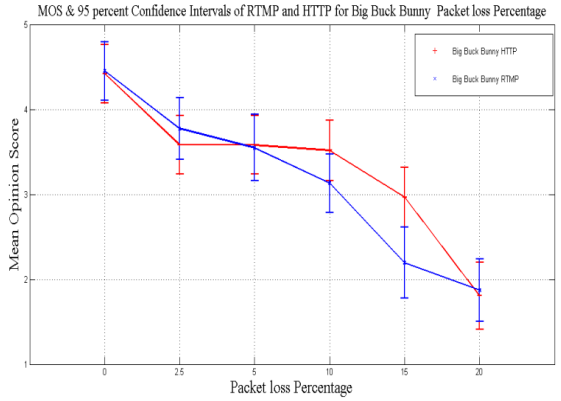
3.4.2 RTMP e protocolli HTTP

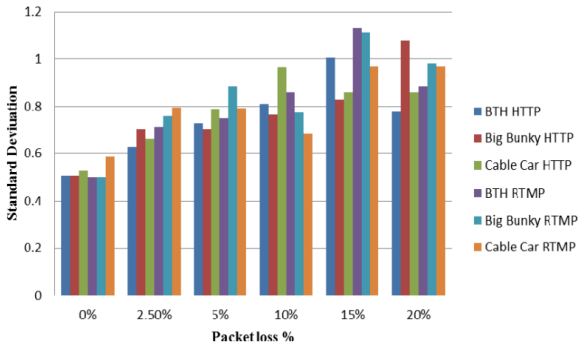
In uno studio [23] gli autori hanno analizzato l'impatto dei parametri di rete, come la perdita di pacchetti e la variazione del ritardo dei pacchetti, per i protocolli RTMP e HTTP, trasmettendo tre diverse sequenze video su una rete controllata. I video raccolti presso il client

sono stati poi valutati utilizzando una valutazione soggettiva, il MOS (Mean Opinion Score), seguendo lo standard dettato dall'International Telecommunication Union (ITU).



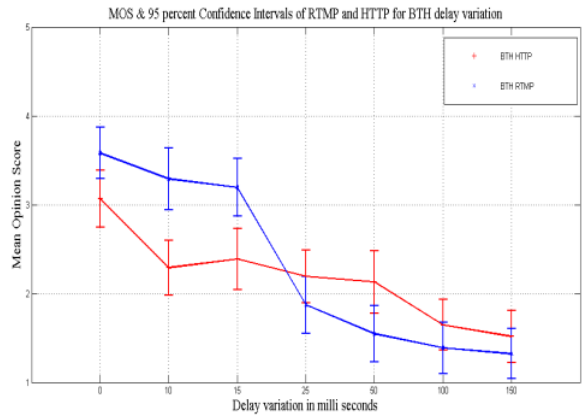


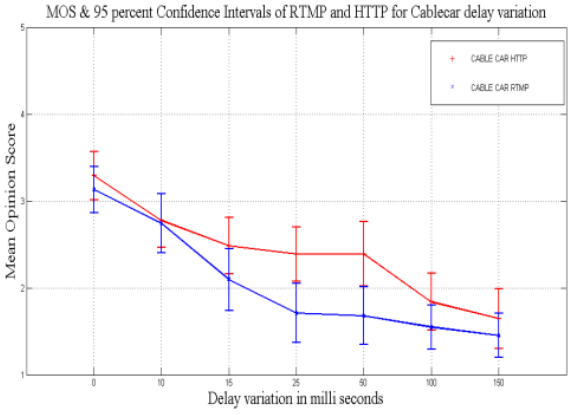


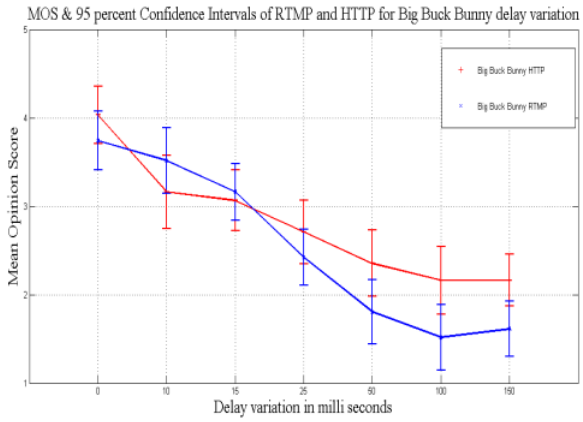


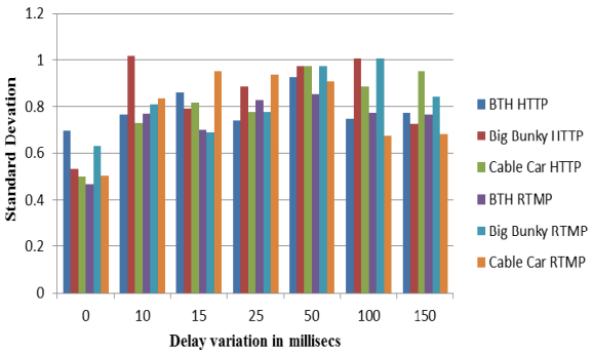
PacketLoss %	BTH HTTP	BTH RTMP	Cable Car HTTP	Cable Car RTMP	Big Buck Bunny HTTP	Big Buck Bunny RTMP
0%	4.433	4.6	4	4.0666	4.56	4.6
2.5%	3.7666	3.9	3.1	3.3	3.7	3.9
5%	3.7666	3.7	3	3.166	3.7	3.666
10%	3.633	3.533	2.8	3.133	3.633	3.23
15%	2.7666	2.633	2.466	2.033	3.066	2.26
20%	1.533	1.666	1.566	1.633	1.866	1.933

Table 4.3.1: MOS Ratings for Packet Loss









Delay Variation (ms)	BTH HTTP	BTH RTMP	Cable Car HTTP	Cable Car RTMP	Big Buck Bunny HTTP	Big Buck Bunny RTMP
150±0	3.166	3.7	3.4	3.3233	4.166	3.866
150±10	2.366	3.4	2.866	2.833	3.266	3.633
150±15	2.466	3.3	2.5666	2.166	3.166	3.266
150±25	2.266	1.933	2.4666	1.766	2.8	2.5
150±50	2.2	1.6	2.4666	1.7333	2.433	1.866
150±100	1.7	1.433	1.9	1.6	2.233	1.566
150±150	1.5	1.366	1.7	1.5	2.233	1.666

Table 4.4.1: MOS Ratings for Jitter Variation

In seguito ai test effettuati gli autori concludono che:

- il protocollo RTMP ha una valutazione migliore di HTTP dallo 0% al 5% di perdita di pacchetti, mentre con l'aumentare della perdita di pacchetti HTTP è leggermente migliore rispetto a RTMP.
- quando la variazione del jitter è molto bassa, gli utenti danno valutazioni migliori per i video in streaming RTMP rispetto alle sequenze video HTTP. Quando si applicano variazioni di jitter elevate, HTTP si comporta meglio di RTMP.

3.4.3 RTMP e SRT

Un altro paper [24] si concentra su un confronto tra i protocolli RTMP e SRT da un punto di vista di latenza end-to-end e di massimo bitrate trasferibile in reti pubbliche.
Per studiare la latenza gli autori hanno misurato il ritardo di uno stream inviato a server sparsi in diverse località utilizzando i due diversi protocolli, i risultati sono riportati nel grafico seguente.

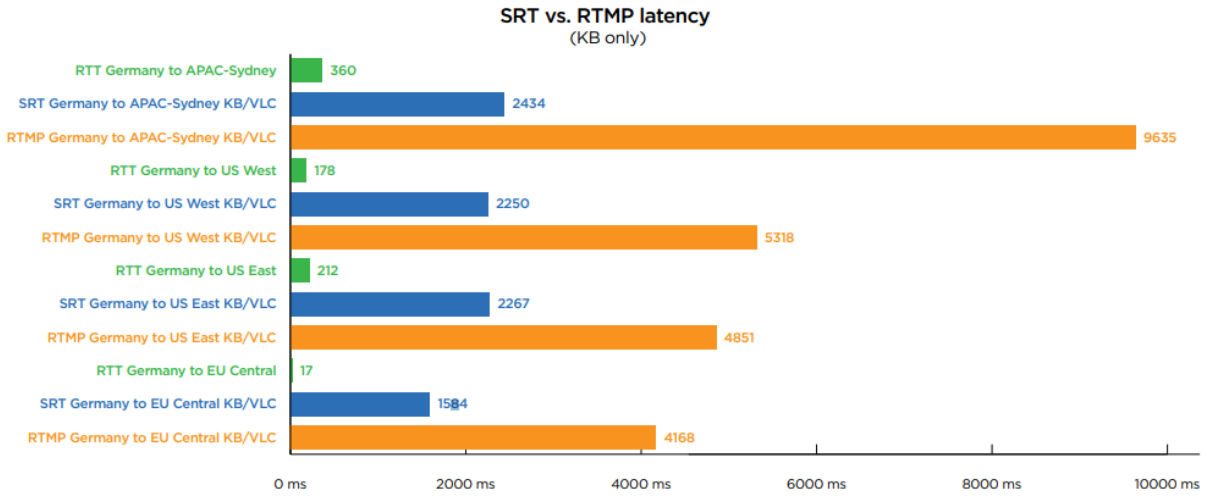


Figura 30 - Confronto latenza tra SRT e RTMP

Per testare il massimo bitrate trasferibile gli autori hanno scelto una location con una buona connessione a Internet, i Microsoft Production Studios di Redmond, Washington. Il nodo internet più vicino si trova a soli 2 hops di distanza e tutti i dispositivi collegati avevano una connettività a 1 Gbps a internet.

Le dimensioni del buffer sono state testate con un flusso a 2 Mbps. Una volta stabili, sono stati condotti test con un flusso a 1, 2, 6, 10 e 20 Mbps.

I risultati hanno mostrato come SRT non ha riscontrato problemi nello streaming fino a 20 Mbps in qualsiasi regione del mondo. RTMP ha funzionato bene quando il mittente e il destinatario si trovavano nello stesso continente, in questo caso il Nord America. Da Redmond è stato possibile inviare fino a 20 Mbps con RTMP in California o in Virginia. Lo streaming verso l'Europa e l'Australia, invece, non era possibile a velocità superiori a 2 Mbps.

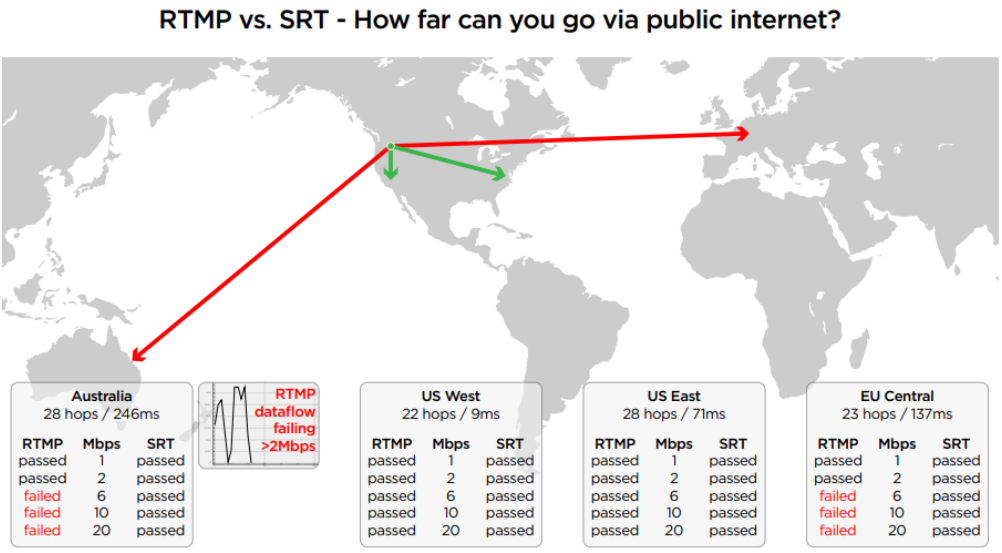


Figura 31 - Confronto RTMP e SRT su streaming a differenti distanze

Da questi test è possibile valutare come in termini di latenza end-to-end e di bitrate massimo trasferibile, SRT supera il protocollo RTMP quando utilizzato nelle reti pubbliche.

3.4.4 Preferenze di utilizzo:

Dal 2021 Video Streaming Latency Report di Wowza [25] è possibile osservare quali protocolli sono i più utilizzati dai vari provider del settore sia per quanto riguarda l'aspetto di ingest, sia per quello di egress.

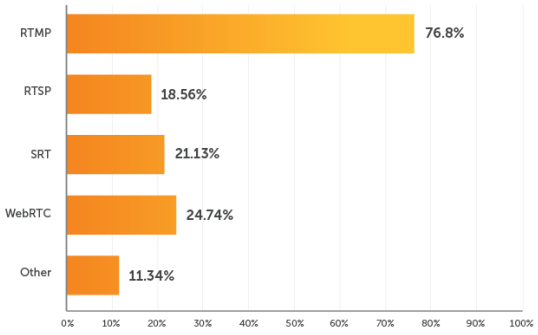


Figura 32 - Preferenze di utilizzo come protocolli di Ingest

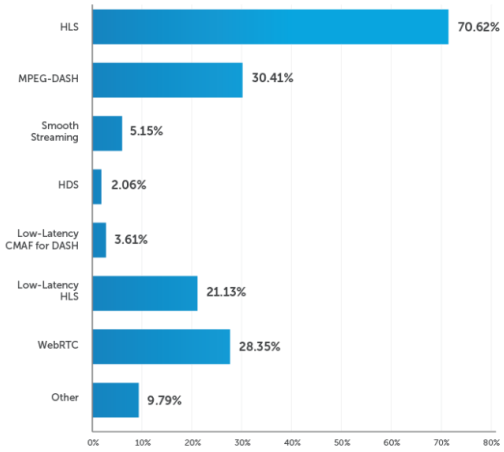


Figura 33 - Preferenze di utilizzo come protocolli di Egress/Delivery

3.5 Cenni a nuove tecniche e strategie per il Live Streaming:

3.5.1 Neural-Enhanced Live Streaming: Improving Live Video Ingest via Online Learning

Questo studio pubblicato nel 2020 [26] mostra come concentrandosi sulla componente di ingest in un flusso live streaming possa migliorare la Quality of Experience (QoE) di tutti gli spettatori connessi. Per farlo viene progettato LiveNAS, ovvero un framework per l'ingest di video in diretta che sfrutta reti neurali sul flusso dati originale per incrementarne la risoluzione, e tramite tecniche di apprendimento online poter massimizzare il guadagno di qualità allocando dinamicamente l'utilizzo delle risorse rispetto al miglioramento della qualità in tempo reale.

3.5.2 Bandwidth Prediction in Low-Latency Chunked Streaming

In questo paper [27] viene presentato ACTE: ABR for Chunked Transfer Encoding, un nuovo schema per le trasmissioni a bitrate adattivo che sfrutta i download segmentizzati dei protocolli basati su HTTP per poter ridurre la latenza del live streaming.
ACTE esegue un'accurata misurazione e previsione del throughput in streaming a bassa latenza e si basa su tre componenti principali:

1. Misurazione del throughput: implementa un metodo di media mobile basato su una finestra scorrevole per misurare la larghezza di banda a livello di chunk.
2. Previsione della larghezza di banda: implementa un filtro lineare adattivo online basato su un algoritmo ricorsivo dei minimi quadrati (RLS) per prevedere la larghezza di banda nel futuro sulla base dei campioni misurati. L'algoritmo RLS funziona bene perché le misure della larghezza di banda sono correlate, stazionarie e deterministiche.
3. Controller ABR: implementa una logica di selezione del bitrate che prende in ingresso i valori di previsione del throughput e calcola il bitrate migliore da selezionare.

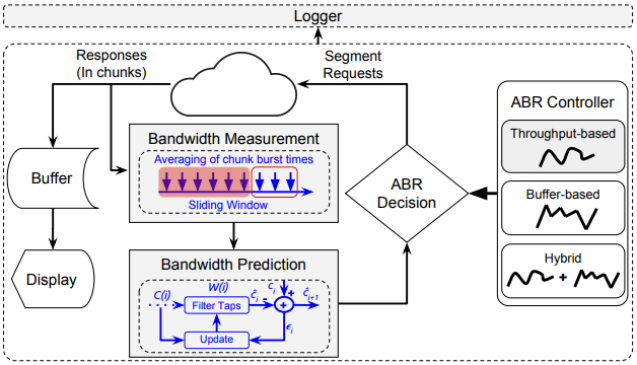


Figura 34 - I componenti di ACTE

Capitolo 4

ARCHITETTURE PER IL LIVE STREAMING

Concettualmente il Live Streaming si può suddividere in quattro componenti: la creazione del contenuto multimediale, l'elaborazione dei dati, la distribuzione dello streaming dati e la riproduzione dello streaming.

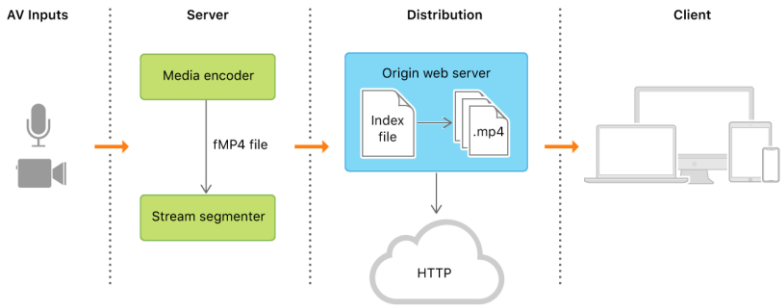


Figura 35 - Esempio di architettura streaming con protocollo HTTP

4.1 Creazione del contenuto:

La creazione del contenuto multimediale avviene tramite una sorgente che crea il flusso audio e video, può essere ad esempio una telecamera, un microfono, o un altro tipo di sorgente multimediale. La sorgente genera il flusso video originale (RAW) che viene indirizzato al componente successivo della catena.

4.2 Elaborazione:

La fase di elaborazione dello streaming solitamente inizia già nel dispositivo sorgente, dove tramite un codificatore software o hardware si procede applicando algoritmi di compressione ai dati in uscita dalla sorgente allo scopo di ridurre l'occupazione di banda. Inoltre si suddivide in pacchetti il flusso per ovviare alla velocità non costante della rete, infatti ogni pacchetto verrà dotato di un timestamp che poi il ricevitore potrà utilizzare per riordinare i pacchetti ricevuti in ordine sparso. L'encoder si occuperà anche di trasmettere lo streaming al server di elaborazione tramite un protocollo di ingest (come ad esempio RTMP).
Il server di elaborazione ha il compito di applicare la transcodifica allo streaming in modo da creare più flussi video a risoluzioni e bitrate differenti in modo da rendere possibile una trasmissione con Adaptive Bitrate (ABR) e poter fornire all'occorrenza al client flussi a minore occupazione di banda per impedire eventuali blocchi nello streaming. Il server si occuperà inoltre della trasmissione tramite un protocollo di egress (ad esempio RTMP, HLS, MPEG-DASH) alla rete di distribuzione.
Un'altra operazione che il server di elaborazione può compiere è inviare il flusso streaming ad un altro server di elaborazione di un'altra piattaforma streaming rendendo in questo modo possibile il restream su più servizi utilizzando una sola connessione d'uscita per la sorgente.

4.3 Distribuzione:

La distribuzione dello streaming dati avviene oggi principalmente per mezzo di Content Delivery Network (CDN) costruite in tutto il mondo da operatori specializzati del settore, che

rendono possibile scalare le operazioni di streaming con facilità e di abbattere il ritardo dei contenuti in diretta.

4.3.1 Content Delivery Network:

Con Content Delivery Network si fa riferimento ad un gruppo di server distribuiti geograficamente in punti chiave della rete internet che collaborano per fornire una rapida trasmissione dei contenuti internet. Oltre ad aumentare la velocità di accesso a determinati contenuti, una CDN può fornire anche un servizio di sicurezza aggiuntivo contribuendo a proteggere da diversi attacchi dannosi comuni, come gli attacchi Distributed Denial of Service (DDoS).

Nella sua essenza, una CDN è una rete di server collegati tra loro con lo scopo di trasmettere contenuti nel modo più rapido, economico, affidabile e sicuro possibile. Per migliorare la velocità e la connettività, una CDN piazzerà dei server presso i punti di scambio tra le diverse reti. Questi punti di interscambio Internet (IXP) sono le sedi principali in cui diversi provider di Internet si connettono per fornire accesso reciproco al traffico che proviene dalle diverse reti. Avendo una connessione a questi punti altamente connessi e ad alta velocità, un provider di CDN è in grado di ridurre i costi e i tempi di transizione nella trasmissione di dati ad alta velocità.

Ad esempio, se immaginiamo un servizio di streaming con server d'origine e di elaborazione in California, e un client che si vuole connettere dall'Italia, per ogni richiesta di pacchetto il client dovrà attendere la creazione di una connessione con il server di origine e inizializzare una nuova trasmissione. Con l'utilizzo di un server edge di una CDN queste operazioni possono essere svolte ad una distanza geograficamente molto inferiore, abbattendo così le attese dovute alla distanza e agli snodi di rete, sarà il server della CDN a instaurare una connessione unica con il server d'origine e a reindirizzare i contenuti a tutti gli utenti della zona contribuendo così ad abbattere non solo le latenze ma anche il carico di lavoro del server d'origine. Inoltre una CDN memorizza nella cache ogni segmento video di uno streaming live e può quindi fornire i segmenti dalla cache invece di ottenere i dati dal server di origine.

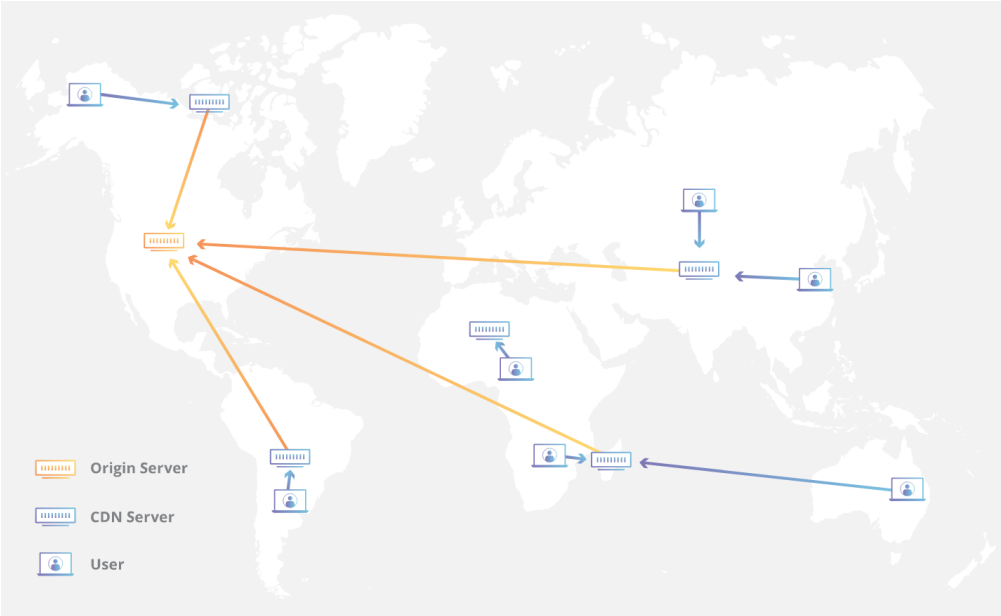


Figura 36 - Schema di collegamento tra utenti, CDN e server di origine

4.4 Riproduzione:

La riproduzione dello streaming avviene per mezzo o di media player lato client, che funzionano nativamente nel dispositivo dello spettatore, o media player integrati all'interno di una pagina web di un service provider (come, ad esempio, i player di YouTube e Facebook), ma è possibile anche sfruttare media player di terze parti per integrare il live streaming nel proprio sito, configurando manualmente la connessione con il server di streaming.
Il software client è responsabile della determinazione dei media appropriati da richiedere, del download di tali risorse e del loro riassemblaggio in modo che i supporti possano essere presentati all'utente in un flusso continuo.

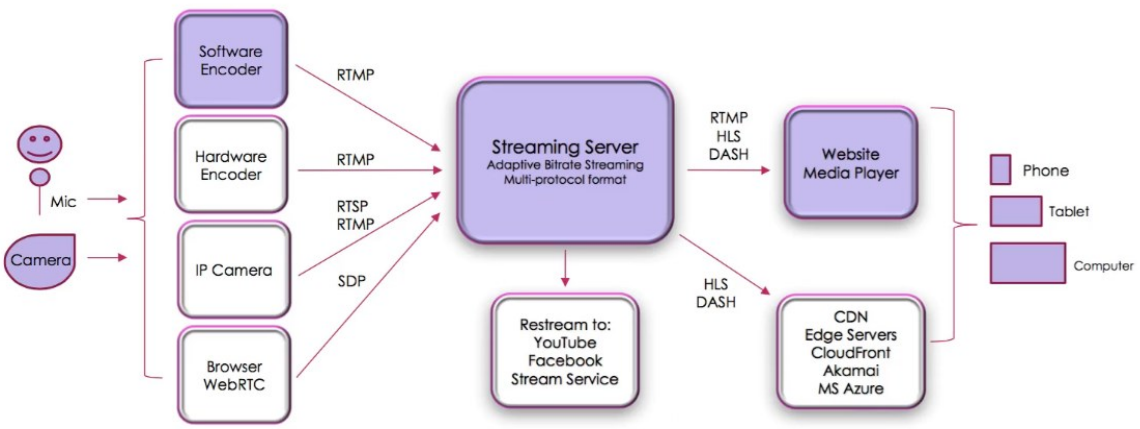


Figura 37 - Architettura per il live streaming

Capitolo 5

CONCLUSIONE

In questa tesi è stato esplorato il mondo del live streaming, dei suoi protocolli e della sua architettura, partendo da un'analisi qualitativa e tecnica delle principali metriche con cui viene analizzato un flusso multimediale in diretta. Nel prosieguo si è continuato esaminando alcuni dei principali protocolli di streaming di uso odierno o in arrivo nel prossimo futuro, mettendoli a confronto in situazioni pratiche attraverso analisi effettuate da diversi studi. Infine, nell'ultimo capitolo, è stata descritta l'architettura cardine del live streaming attraverso la rete internet e lo sfruttamento della content delivery network.

Dai protocolli esaminati risulta evidente come la tematica della qualità percepita all'utente finale, nel ruolo della QoE, sia la principale direzione intrapresa per il miglioramento dello streaming in diretta. Difatti la latenza, l'adattamento video e i confronti tra diversi protocolli sono stati analizzati nel dettaglio mostrando come la tecnica dell'adaptive bitrate sia stata sfruttata ampiamente in combinazione con i protocolli HTTP per inibire il più possibile la possibilità di stallo, fattore principale di degrado nell'esperienza utente, mentre l'architettura di rete nella forma della content delivery network è utilizzata per distribuire i flussi video abbassando la latenza e mitigando la congestione di rete.

Questi aspetti, possiamo immaginare, rappresenteranno elementi importanti anche nello sviluppo di protocolli di nuova generazione con implementazioni sempre più intelligenti dello streaming adattivo e che possano fornire robuste tecniche per la diminuzione degli stalli.

RIFERIMENTI:

[1] “On-Demand Viewing Growing Much Faster Than Live, Says Conviva”. Troy Dreier - https://www.streamingmedia.com/Articles/News/Online-Video-News/On-Demand-Viewing-Growing-Much-Faster-Than-Live-Says-Conviva-133418.aspx

[2] “Live Video Streaming: A Global Perspective”. Iab.com - https://www.iab.com/insights/live-video-streaming-2018/

[3] “GWI - Coronavirus Research | March 2020” - Gwi.com - https://www.gwi.com/hubfs/1.%20Coronavirus%20Research%20PDFs/GWI%20coronavirus%20findings%20March%202020%20-%20Multi-Market%20data%20(Release%203).pdf

[4] “B2C Content Marketing 2020: Benchmarks, Budgets, and Trends”. Content marketing institute - https://contentmarketinginstitute.com/wp-content/uploads/2019/12/2020_B2C_Research_Final.pdf

[5] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld and P. Tran-Gia, "A Survey on Quality of Experience of HTTP Adaptive Streaming," in IEEE Communications Surveys & Tutorials, vol. 17, no. 1, pp. 469-492, Firstquarter 2015, https://doi.org/10.1109/COMST.2014.2360940

[6] T. Hossfeld, S. Egger, R. Schatz, M. Fiedler, K. Masuch and C. Lorentzen, "Initial delay vs. interruptions: Between the devil and the deep blue sea," 2012 Fourth International Workshop on Quality of Multimedia Experience, 2012, pp. 1-6, doi: https://doi.org/10.1109/QoMEX.2012.6263849

[7] Alessandro Finamore, Marco Mellia, Maurizio M. Munafò, Ruben Torres, and Sanjay G. Rao. 2011. YouTube everywhere: impact of device and infrastructure synergies on user experience. In Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference (IMC '11). Association for Computing Machinery, New York, NY, USA, 345–360. https://doi.org/10.1145/2068816.2068849

[8] L. Chen, Y. Zhou and D. M. Chiu, "Video Browsing - A Study of User Behavior in Online VoD Services," 2013 22nd International Conference on Computer Communication and Networks (ICCCN), 2013, pp. 1-7, https://doi.org/10.1109/ICCCN.2013.6614209

[9] Yao, J., Kanhere, S.S., Hossain, I., Hassan, M. (2011). Empirical Evaluation of HTTP Adaptive Streaming under Vehicular Mobility. In: Domingo-Pascual, J., Manzoni, P., Palazzo, S., Pont, A., Scoglio, C. (eds) NETWORKING 2011. NETWORKING 2011. Lecture Notes in Computer Science, vol 6640. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-20757-0_8

[10] T. Zinner, T. Hossfeld, T. N. Minhas, and M. Fiedler, 'Controlled vs. Uncontrolled Degradations of QoE : The Provisioning-Delivery Hysteresis in Case of Video', presented at the EuroITV 2010 Workshop: Quality of Experience for Multimedia Content Sharing, 2010, Published. https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A835273&dswid=5759

[11] Knoche, H., McCarthy, J.D. & Sasse, M.A. How low can you go? The effect of low resolutions on shot types in mobile TV. Multimed Tools Appl 36, 145–166 (2008). https://doi.org/10.1007/s11042-006-0076-5

[12] Pastrana-Vidal, Ricardo & Gicquel, Jean & Colomes, Catherine & Cherifi, Hocine. (2004). Sporadic frame dropping impact on quality perception. Proceedings of SPIE - The International Society for Optical Engineering. 5292. https://doi.org/10.1117/12.525746

[13] G. Ghinea and J. P. Thomas. 1998. QoS impact on user perception and understanding of multimedia video clips. In Proceedings of the sixth ACM international conference on Multimedia (MULTIMEDIA '98). Association for Computing Machinery, New York, NY, USA, 49–54. https://doi.org/10.1145/290747.290754

[14] "New Study: Quality of OTT Video Streaming Experiences Directly Tied to Viewer Loyalty, Service Provider Success". Akamai Technologies, Inc. - https://www.prnewswire.com/news-releases/new-study-quality-of-ott-video-streaming-experiences-directly-tied-to-viewer-loyalty-service-provider-success-300475918.html

[15] J. Kua, G. Armitage and P. Branch, "A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming Over HTTP," in IEEE Communications Surveys & Tutorials, vol. 19, no. 3, pp. 1842-1866, thirdquarter 2017, https://doi.org/10.1109/COMST.2017.2685630

[16] "What Is Low Latency and Who Needs It?". Wowza Media Systems - https://www.wowza.com/blog/what-is-low-latency-and-who-needs-it

[17] "QoS monitoring in OTT: importance, challenges & best practices". Brenton Ough - https://blog.touchstream.media/qos-monitoring

[18] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer and R. Zimmermann, "A Survey on Bitrate Adaptation Schemes for Streaming Media Over HTTP," in IEEE Communications Surveys & Tutorials, vol. 21, no. 1, pp. 562-585, Firstquarter 2019, https://doi.org/10.1109/COMST.2018.2862938

[19] T. Mangla, E. Halepovic, M. Ammar and E. Zegura, "eMIMIC: Estimating HTTP-Based Video QoE Metrics from Encrypted Network Traffic," 2018 Network Traffic Measurement and Analysis Conference (TMA), 2018, pp. 1-8, https://doi.org/10.23919/TMA.2018.8506519

[20] Poretsky, S., Perser, J., Erramilli, S., and S. Khurana, "Terminology for Benchmarking Network-layer Traffic Control Mechanisms", RFC 4689, DOI 10.17487/RFC4689, October 2006, https://www.rfc-editor.org/info/rfc4689

[21] Yunhong Gu, “UDT: UDP-based Data Transfer Protocol” April 2010, https://datatracker.ietf.org/doc/pdf/draft-gg-udt-03

[22] L. Nuñez and R. M. Toasa, "Performance evaluation of RTMP, RTSP and HLS protocols for IPTV in mobile networks," 2020 15th Iberian Conference on Information Systems and Technologies (CISTI), 2020, pp. 1-5, https://ieeexplore.ieee.org/document/9140848

[23] R. G. Guniganti and S. Ankam, ‘A Comparison of RTMP and HTTP Protocols with respect to Packet Loss and Delay Variation based on QoE’, Dissertation, 2013 - https://www.diva-portal.org/smash/get/diva2:832774/FULLTEXT01.pdf

[24] - “RTMP vs. SRT: Comparing Latency and Maximum Bandwidth”, Haivision - https://www3.haivision.com/e/38322/wp-rtmp/93g6t6/1995741960

[25] “2021 Video Streaming Latency Report”, Wowza Media Systems - https://f.hubspotusercontent20.net/hubfs/229276/2021%20Low%20Latency%20Report/streaming-video-latency-report-interactive-2021.pdf

[26] Jaehong Kim, Youngmok Jung, Hyunho Yeo, Juncheol Ye, and Dongsu Han. 2020. Neural-Enhanced Live Streaming: Improving Live Video Ingest via Online Learning. In Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM '20). Association for Computing Machinery, New York, NY, USA, 107–125. https://doi.org/10.1145/3387514.3405856

[27] Abdelhak Bentaleb, Christian Timmerer, Ali C. Begen, and Roger Zimmermann. 2019. Bandwidth prediction in low-latency chunked streaming. In Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '19). Association for Computing Machinery, New York, NY, USA, 7–13. https://doi.org/10.1145/3304112.3325611

[28] Adobe’s Real Time Messaging Protocol, archiviato in data 05/04/2016 - https://web.archive.org/web/20160405144013/https://wwwimages2.adobe.com/content/dam/Adobe/en/devnet/rtmp/pdf/rtmp_specification_1.0.pdf

[29] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, https://www.rfc-editor.org/info/rfc3550

[30] Schulzrinne, H., Rao, A., Lanphier, R., Westerlund, M., and M. Stiemerling, Ed., "Real-Time Streaming Protocol Version 2.0", RFC 7826, DOI 10.17487/RFC7826, December 2016, https://www.rfc-editor.org/info/rfc7826

[31] Pantos, R., Ed., and W. May, "HTTP Live Streaming", RFC 8216, DOI 10.17487/RFC8216, August 2017, https://www.rfc-editor.org/info/rfc8216

[33] “SRT Protocol Technical Overview”, Haivision - https://www.haivision.com/resources/white-paper/srt-protocol-technical-overview/

[34] Loreto S., Pietro Romano S., Real-Time Communication with WebRTC: Peer-to-Peer in the Browser, O'Reilly Media, 2014

[35] “Reliable Internet Stream Transport”, Videoservicesforum.org - https://www.videoservicesforum.org/activity_groups/RIST_poster_for_VidTrans2018Feb25.pdf