POLITECNICO DI TORINO

Corso di Laurea Magistrale in
Ingegneria Informatica (Computer Engineering)

Tesi di Laurea Magistrale

Un'applicazione distribuita business-critical per la trasmissione end-to-end resiliente di flussi multimediali in ambienti WiFi critici

Relatore

Prof. Giovanni MALNATI

Candidato

Matteo BUFFO

Luglio 2021

Ringraziamenti

Desidero innanzitutto ringraziare il prof. Malnati, relatore di questo lavoro di tesi, per la sua disponibilità e professionalità. Ringrazio i futuri colleghi di RCS ETM Sicurezza Spa, con la certezza di lavorare presto insieme, e in particolar modo Lorenzo, il mio formadibile supervisore aziendale, che mi ha seguito con i suoi indispensabili suggerimenti fino alla stesura di questa tesi.

Un caloroso ringraziamento va ai miei amici, vicini e lontani, con cui ho trascorso esperienze uniche e indimenticabili. Grazie di cuore anche ai miei colleghi universitari, che nel corso di questi anni di studi al Politecnico di Torino hanno condiviso con me ansie, delusioni, risate, gioie e successi. A voi tutti, ovunque vi troviate, giunga il mio più sincero e affettuoso “in bocca al lupo” per il vostro futuro. Ad maiora semper!

Dedico infine questo importante traguardo alla mia famiglia, sempre unita e al mio fianco sia nei momenti belli che in quelli brutti della mia vita: perché nulla potrà mai dividerci davvero. A mio padre, che con la sua inesauribile energia mi ha spronato a migliorarmi e a dare sempre il massimo. A mia madre, per la sua infinita pazienza e l’affetto senza pari che mi ha dimostrato in tutti questi anni. A mio fratello Giulio, l’ormai cresciuto "bambino saggio" che mi è stato vicino con i suoi preziosissimi consigli. A mio fratello Giorgio, che con il suo entusiasmo riesce a rallegrare e rendere speciali le mie giornate.

Matteo

Sommario

Contesto

Il lavoro presentato in questa tesi di Laurea Magistrale è stato svolto presso RCS ETM Sicurezza Spa, un'azienda con sede a Milano e attiva dal 1993 nel settore delle Lawful Interception. Con questo termine si indica un processo in cui un fornitore di servizi o un operatore di rete intercetta le telecomunicazioni di una determinata persona o organizzazione e le trasmette ad un ente legalmente autorizzato, come le Forze dell'Ordine o agenzie investigative internazionali.

Le Lawful Interception si distinguono in intercettazioni telefoniche, telematiche e ambientali. Queste ultime sono realizzate con l'impiego di microspie ambientali, dei dispositivi elettronici che vengono posizionati dagli addetti alle intercettazioni in punti celati o poco accessibili all'interno degli ambienti da monitorare. In questo modo le microspie catturano, elaborano e trasmettono in rete i suoni e le immagini richiesti dalle Forze dell'Ordine.

Nel contesto delle intercettazioni ambientali, una delle sfide più impegnative per gli operatori al giorno d'oggi è rappresentata dagli ambienti WiFi critici, ossia ambienti caratterizzati da scarsa copertura di rete o dalla presenza di barriere architettoniche che compromettono la connettività tra i dispositivi trasmittenti e i ricevitori. Di conseguenza, gli investigatori non hanno il pieno controllo né sullo stato della rete né sullo stato delle microspie.

Obiettivi

Questa tesi si pone come obiettivi la progettazione e lo sviluppo di un'applicazione web distribuita, da impiegare nel contesto delle Lawful Interception in ambienti a connettività di rete limitata, fondata sulla trasmissione resiliente di flussi contenenti audio, video e dati in generale.

Metodo

Nel capitolo 2 si analizza lo stato dell'arte della soluzione attualmente adottata per l'attuazione delle intercettazioni ambientali. Si introduce dapprima il processo di live streaming e il suo legame con le Lawful Interception, dopodiché si elencano e descrivono le fasi che lo compongono: cattura di un evento, compressione, frammentazione, conversione e distribuzione del flusso multimediale e, infine, riproduzione del contenuto sui dispositivi utente. Si passano dunque in rassegna gli svantaggi delle scelte progettuali odierne relative alle varie fasi del processo attraverso la descrizione di un esempio reale.

Nel capitolo 3 si propone, alla luce delle criticità presentate nello stato dell'arte, una nuova soluzione basata su MPEG-Transport Stream (MPEG-TS) e HTTP Live Streaming (HLS) e si elencano i vantaggi di questi strumenti. Il primo è un formato contenitore sviluppato dal comitato MPEG e impiegato nella frammentazione del flusso multimediale. Il secondo è un protocollo, sviluppato da Apple, per la distribuzione dei dati in rete ed è considerato al giorno d'oggi lo standard de facto per lo streaming multimediale a bitrate adattivo.

Nel capitolo 4 si affronta la fase di analisi di fattibilità. Dopo una breve panoramica delle librerie HLS, si mostra il funzionamento di base della libreria JavaScript+TypeScript hls.js per creare un player HLS all'interno di una pagina HTML di prova. Successivamente, si esamina in maniera approfondita l'architettura della libreria e si descrivono i contributi apportati al fine di separare le tracce audio e video del flusso, indirizzarle verso elementi HTML dedicati e introdurre il canale dati per la trasmissione di informazioni definite dall'utente. Il capitolo prosegue con la fase di integrazione. Viene presentato il player aziendale, un'applicazione web creata con i framework React e Angular che riproduce contenuti esclusivamente audio.

Si descrivono dunque i passaggi per integrare i canali video e dati, nonché alcune funzionalità utili, come l'applicazione di filtri visivi tramite la libreria JavaScript glfx.js e la cattura dei fotogrammi del video.

Nel capitolo 5 si espongono i possibili sviluppi futuri sul player multimediale. Dopo aver definito la computer vision e il ruolo che essa riveste nelle applicazioni odierne, si evidenzia l'utilità dell'impiego di algoritmi di object detection e recognition nel campo delle Lawful Interception ambientali. Dopodiché si passa in rassegna il funzionamento di face-api.js, una libreria JavaScript per il riconoscimento facciale, e infine si fornisce una disamina delle soluzioni per l'implementazione dell'object detection e recognition all'interno dell'applicazione.

Indice

Elenco delle figure

Elenco dei listati

Elenco delle tabelle

Abbreviazioni

A/V

audio/video

A/V/D

audio/video/dati

AAC

Advanced Audio Coding

AAC-LC

AAC Low Complexity

ADMF

Administration Function

AES

Advanced Encryption System

AO

Authorized Organization

AP

Access Provider

API

Application Programming Interface

AVI

Audio Video Interleave

AWS

Amazon Web Services

CD

Call Data

CDN

Content Delivery Network

CSP

Communications Service Provider

CSS

Cascading Style Sheets

DASH

Dynamic Adaptive Streaming over HTTP

DB

Database

DOM

Document Object Model

DVD

Digital Versatile Disk

ECMA

European Computer Manufacturers Association

EME

Encrypted Media Extensions

ETSI

European Telecommunications Standards Institute

GPS

Global Positioning System

HI

Handover Interface

HLS

HTTP Live Streaming

HTML

HyperText Markup Language

HTTP

HyperText Transfer Protocol

IA

Intelligenza Artificiale

IEC

International Electrotechnical Commission

IETF

Internet Engineering Task Force

IRI

Intercept Related Information

ISO

International Organization for Standardization

LEMF

Law Enforcement Monitoring Facility

LI

Lawful Interception

LL-HLS

Low-Latency HLS

MDF

Mediation and Delivery Function

MPEG

Moving Picture Experts Group

MSE

Media Source Extensions

npm

Node Package Manager

NWO

Network Operator

PES

Packetized Elementary Streams

PiP

Picture-in-Picture

POI

Point of Interception

PS

Program Stream

RD

Retained Data

RDHI

Retained Data Handover Interface

RFC

Request for Comments

RTMP

Real-Time Messaging Protocol

RTSP

Real-Time Streaming Protocol

SMS

Short Message Service

SvP

Service Provider

TS

Transport Stream

UDP

User Datagram Protocol

UI

User Interface

UMTS

Universal Mobile Telecommunications System

URI

Unified Resource Identifier

URL

Uniform Resource Locator

UTF

Unicode Transformation Format

VHF

Very High Frequency

VoIP

Voice over IP

VU

Volume Units

W3C

World Wide Web Consortium

WebGL

Web-based Graphics Library

WWDC

WorldWide Developers Conference

yarn

Yet Another Resource Negotiator

Capitolo 1

Introduzione

1.1 Lawful Interception

1.1.1 Definizione

Con il termine Lawful Interception (LI) si intende quel processo per il quale un fornitore di servizi o un operatore di rete intercetta le telecomunicazioni di una determinata persona o organizzazione e le trasmette ad un servizio legalmente autorizzato.

L'attuazione della LI è richiesta dalla Risoluzione del Consiglio dell'Unione europea del 1995, che rende quest'attività uno strumento per accertare ipotesi criminose e prevenire i reati, tra cui le truffe, intese in molteplici accezioni, ed atti terroristici [1]. L'European Telecommunications Standards Institute (ETSI) è uno degli organismi leader per la standardizzazione a livello globale dei concetti e del funzionamento delle LI, a cui hanno contribuito aziende come Cisco e l'Internet Engineering Task Force (IETF).

La LI ha lo scopo di raccogliere i dati provenienti dalla rete di telecomunicazione per svolgere delle analisi e, nel contesto giudiziario, rappresenta un mezzo di ricerca della prova [2]. I dati raccolti durante le LI, denominati Call Data (CD) negli Stati Uniti e Intercept Related Information (IRI) in Europa, possono riguardare sia il contenuto delle comunicazioni (suoni, voci, conversazioni in formato audio e/o video) che le informazioni relative allo stato della rete ed alla sua gestione. Se i dati

non vengono raccolti in tempo reale, essi vengono chiamati Retained Data (RD) [3].

1.1.2 Caratteristiche

L'attività delle LI vede coinvolte due parti:

- il Communications Service Provider (CSP), che indica un'organizzazione (ad esempio un Service Provider (SvP), un Network Operator (NWO) o un Access Provider (AP)) obbligata per legge a fornire il materiale raccolto durante le intercettazioni [4];
- l'Authorized Organization (AO), che rappresenta un ente (ad esempio i reparti investigativi delle Forze dell'Ordine, un'agenzia investigativa privata o i servizi di intelligence) autorizzato dalla legge a richiedere e ricevere i dati [5, p. 12].

Come mostrato in figura 1.1, esse comunicano attraverso due interfacce fisiche e logiche dette Handover Interface (HI):

- sull'Handover Interface A (HI-A), bidirezionale, l'AO invia le richieste di informazioni al CSP e riceve le relative risposte;
- sull'Handover Interface B (HI-B), nella maggior parte dei casi unidirezionale, il CSP invia i dati cifrati dell'intercettazione all'AO.

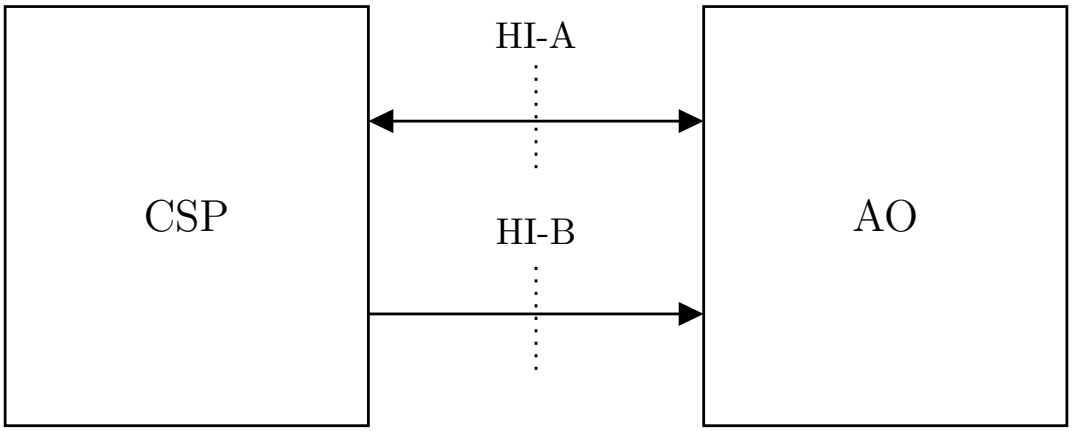

Figura 1.1: Schema di comunicazione tra CSP e AO [5, p. 15].

Facendo riferimento agli schemi delle figure 1.2 e 1.3, all'interno del CSP si definiscono tre funzioni:

- raccolta dati, svolta dai Point of Interception (POI), che ricevono i dati provenienti dalla rete e li elaborano prima di conservarli e inoltrarli. I POI sono rappresentati dai dispositivi trasmittenti impiegati nelle intercettazioni e presentati nel sottoparagrafo 1.2.1;
- gestione della conservazione dei dati, sui quali è possibile eseguire interrogazioni e definire indici. Include la Mediation and Delivery Function (MDF), che coincide con il Network Operator, e la Law Enforcement Monitoring Facility (LEMF), che coincide con il server di ricezione;
- amministrativa, assolta dalla Administration Function (ADMF), per gestire lo scambio di richieste e risposte con l'AO nonché lo stato dei POI e della MDF.

All'interno dell'AO si definiscono due entità:

- un'autorità emittente (Issuing Authority) che invia le richieste Retained Data Handover Interface (RDHI) sull'HI-A;
- un'autorità ricevente (Receiving Authority) che riceve le risposte RDHI sull'HI-B.

Nella maggior parte dei casi, esse coincidono [5, pp. 15–16].

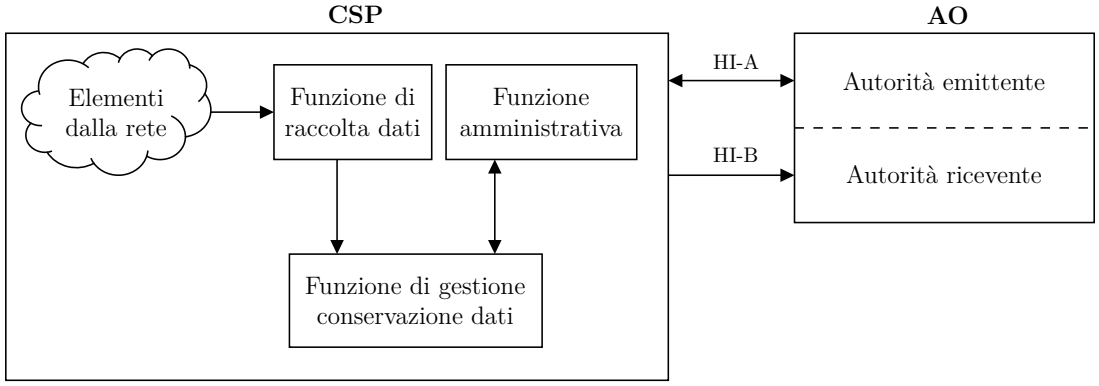

Figura 1.2: Schema dettagliato di comunicazione tra CSP e AO [5, p. 16].

1.1.3 Classificazione

Le intercettazioni in generale possono essere classificate in:

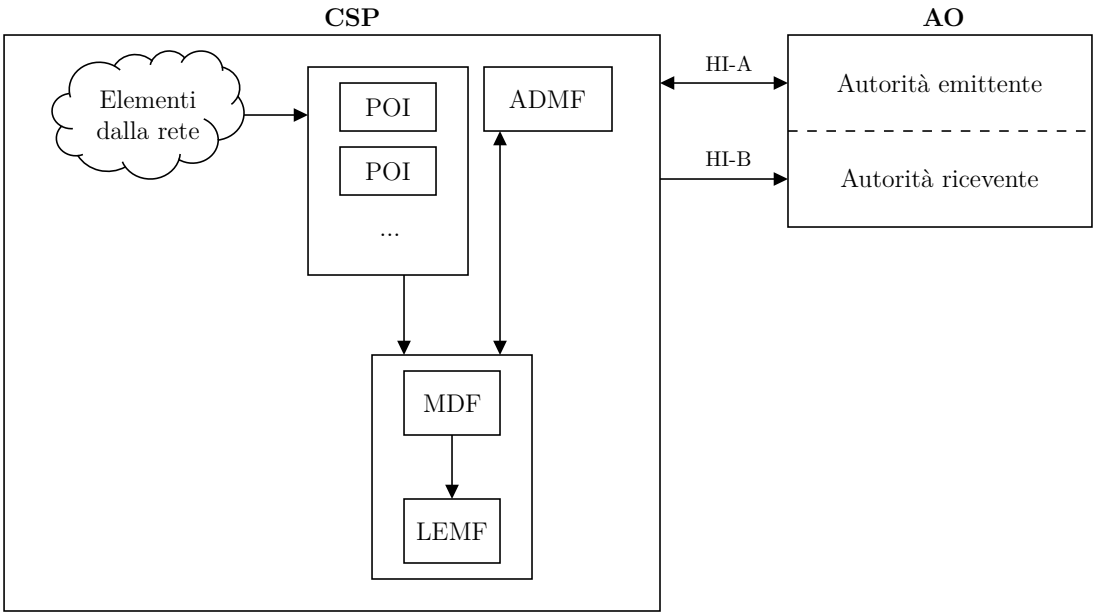

Figura 1.3: Struttura interna del CSP nello schema di comunicazione tra CSP e AO.

- telefoniche, che operano con o senza la collaborazione degli operatori telefonici, obbligati a mettere a disposizione le proprie strutture tecnologiche ed organizzative all'AO. Le linee telefoniche obiettivo dell'intercettazione vengono duplicate, senza che l'utilizzatore se ne accorga, e le conversazioni in copia sono instradate verso un apposito centro intercettazioni, in cui possono essere registrate e conservate. In genere, le registrazioni vengono protette con sistemi di cifratura;

- telematiche, che si basano classicamente sull'acquisizione di pacchetti di dati in transito sulla rete. In questo modo, gli operatori sono in grado di intercettare telefonate basate su VoIP e scambi di SMS, email e file. I dati trasmessi o ricevuti da un singolo utente o da gruppi di utenti vengono:

  – acquisiti mentre transitano dal computer emittente a quello ricevente;

  – trasmessi dal computer emittente tramite appositi software-spia installati all'insaputa dell'utente.

- ambientali, presentate nel dettaglio nel paragrafo 1.2 [2].

Esiste una serie di parametri che consentono agli investigatori di scegliere quali tipologie di LI sia necessario svolgere per le indagini. Tra questi figurano:

- l'utilizzo o meno del telefono cellulare come mezzo di comunicazione da parte di un soggetto indagato;
- la segretezza delle indagini;
- le eventuali contromisure adottate contro le intercettazioni.

1.2 LI ambientali

1.2.1 Definizione

Le intercettazioni ambientali sono realizzate principalmente con l'impiego di dispositivi elettronici, detti trasmettitori, quali ad esempio:

- microspie, apparecchi di dimensioni ridotte idonei a captare il materiale e trasmetterlo via cavo o via radio a un'idonea stazione ricevente, generalmente lontana dal punto di cattura. Qui, il contenuto captato ed intercettato viene ascoltato o visionato, in funzione della tipologia di media a cui si riferisce, e successivamente trascritto o elaborato;
- telecamere nascoste, che catturano immagini e video in maniera del tutto analoga alle microspie;
- microfoni direzionali, dei particolari microfoni dotati di un cono con funzione di amplificazione selettiva del suono proveniente da una o più direzioni, con i quali è possibile ascoltare a distanza senza avere un dispositivo nei pressi dell'area intercettata;
- microfoni a contatto, dispositivi equipaggiati di software per la massimizzazione della resa fonica in termini volumetrici che tipicamente consentono di captare suoni la cui sorgente si trova al di là di una parete o una soletta.

Inoltre, si distinguono due categorie di microspie:

- cimici (bugs in inglese), definite così per via delle loro dimensioni, che trasmettono esclusivamente audio in bassa qualità su onde radio a Very High Frequency (VHF).

- ambientali, le quali, rispetto alle cimici, sono dotate di tecnologie più avanzate e hanno dimensioni maggiori. Possono trasmettere sia flussi audio/video (A/V) che dati generici (ad esempio quelli relativi alle coordinate GPS). Poiché operano su reti UMTS, garantiscono un raggio d'azione più ampio rispetto a quello raggiunto dalle cimici.

In base agli scopi delle indagini, alle risorse elettroniche a disposizione, all'infrastruttura di rete e, di conseguenza, alla modalità di trasmissione dei dati, gli ambienti in cui avvengono le intercettazioni possono essere distinti come:

- fissi, tipicamente luoghi privati o pubblici come abitazioni, uffici, locali, scuole etc.;

- mobili, rappresentati dai mezzi di trasporto terrestri, acquatici e aerei.

È compito degli investigatori e, in generale, degli operatori di Pubblica Sicurezza, esaminare l'ambiente per individuare dei punti strategici, poco usati e accessibili, in cui nascondere le microspie da utilizzare per l'intercettazione (ad esempio, dietro ad un orologio da parete, dentro un vaso o sotto il sedile di un'automobile). Le cimici, a causa delle limitazioni tecniche intrinseche, vengono posizionate in ambienti prevalentemente fissi.

1.2.2 Ambienti WiFi critici

Nella maggior parte dei casi gli addetti alle LI in ambienti mobili devono poter contare sulla disponibilità del tempo reale, una condizione in cui la correttezza delle operazioni svolte è garantita dai tempi ridotti con cui avviene la trasmissione dei dati dal dispositivo al server di analisi.

Nella maggior parte degli scenari applicativi mobili, gli addetti alle LI devono poter contare sulla disponibilità del tempo reale. La maggioranza delle attività investigative, infatti, vengono condotte per ragioni preventive al fine di individuare e

anticipare azioni fraudolente in corso. Questa necessità rende particolarmente critico l'intero processo attuativo in quanto, in contesti mobili, il tempo reale risulta non garantito.

Si definisce inoltre latenza il ritardo che intercorre tra l'acquisizione di un evento e la sua consegna al ricevitore. Poiché è impossibile raggiungere una condizione di tempo reale ideale con latenza nulla, si cerca di ottenere una latenza sufficientemente bassa. Latenze nell'intervallo 10÷30 s sono in genere accettabili per la maggior parte delle applicazioni. Tuttavia, esistono degli scenari più esigenti per i quali si vogliono - e si devono - ottenere latenze molto più basse (in tal caso con valori compresi tra 30 ms e 10 s). Di questi scenari fanno parte le seguenti categorie, per le quali sono stati riportati alcuni esempi di piattaforme:

- dirette streaming di eventi sportivi come DAZN e eSports come YouTube e Twitch.tv;
- videochiamate/videoconferenze come Google Meet/Hangouts, FaceTime, e Skype;
- aste come Sotheby's e siti per il gioco d'azzardo online come BetOnline;
- LI preventive, che hanno come finalità la prevenzione di pericolosi atti di criminalità organizzata, terrorismo o eversione dell'ordine costituzionale [6].

Sfortunatamente, il prerequisito del tempo reale viene meno se nel corso delle attività investigative il dispositivo di intercettazione si imbatte in ostacoli fisici che compromettono la connettività tra trasmettitori e ricevitori. Tra i casi più tipici si annoverano locali interrati o seminterrati (come cantine, uffici e parcheggi), gallerie e zone isolate o di montagna in cui la copertura di rete è scarsa o addirittura assente.

Alla luce di quanto appena descritto si definiscono ambienti WiFi critici quegli ambienti, prevalentemente mobili, in cui la disponibilità della rete di comunicazione tra le parti coinvolte non è garantita: di conseguenza, gli investigatori non hanno il pieno controllo né della connettività di rete né dello stato dei dispositivi trasmittenti. Inoltre, si definiscono ambienti estremi quegli ambienti critici in cui l'assenza della connettività di rete si protrae per un periodo di tempo prolungato.

Negli scenari descritti, i servizi investigativi non possono garantire alcuna prevenzione di eventuali crimini. Le attività dunque possono essere solo condotte in post-analisi attraverso dati intercettati che i dispositivi coinvolti devono registrare a bordo e trasmettere verso il server di analisi solo quando la disponibilità di rete lo consente. Nell'ambito delle LI, quest'ultimo meccanismo viene indicato come differita.

1.3 Riepilogo

Il compito principale degli investigatori è quello di ricevere in tempo reale il materiale audiovisivo captato dai dispositivi. Tuttavia, alla luce di quanto esposto nel sottoparagrafo 1.2.2, questo compito rappresenta una vera e propria sfida quando si lavora in contesti critici, poiché le condizioni avverse dell'ambiente e della rete compromettono la garanzia del tempo reale e quindi influiscono negativamente sulla buona riuscita dell'operazione. Gli ambienti WiFi critici necessitano dunque l'adozione di strategie diverse per rendere le LI resilienti alle disconnessioni dalla rete e mitigare il problema.

Capitolo 2

Stato dell'arte

2.1 Live streaming

2.1.1 Definizione

Il live streaming è un processo che consiste nella trasmissione di contenuti multimediali in tempo reale ad utenti o spettatori. Esso è un fenomeno la cui evoluzione tecnologica procede di pari passo con quella di Internet e nel particolare periodo storico che stiamo vivendo, in cui le restrizioni dovute alla pandemia di COVID-19 hanno causato un notevole incremento dell'utilizzo dei servizi in rete, ha guadagnato popolarità tra gli utenti del web. Inoltre, le dirette streaming abbracciano numerosi ambiti, dalle piattaforme per lo streaming di competizioni videoludiche (ad esempio Twitch.tv) ai social network (Facebook, Instagram etc.) passando per i servizi dedicati alle vendite di e-commerce, ai concerti online e altresì alle Lawful Interception, oggetto di questa tesi.

2.1.2 Architettura

Il processo di live streaming è composto dai seguenti step:

1. cattura, che consiste nella registrazione dell'audio e del video di un evento mediante un dispositivo elettronico (ad esempio telefoni cellulari, videocamere, webcam, ambientali etc.);

2. compressione, in cui i dati grezzi catturati sono convertiti in un formato di compressione digitale attraverso il codec. Tra i codec più noti si riportano:

   - H.264 e AV1 per il video;
   - AAC e MP3 per l'audio.

   Il codec è un componente fondamentale nel processo di live streaming, poiché deve essere scelto in maniera tale da garantire la distribuzione efficiente del flusso A/V attraverso Internet e la sua riproduzione sui dispositivi di destinazione;

3. frammentazione, in cui il flusso A/V compresso è spezzettato e i segmenti prodotti sono impacchettati in un formato container (detto anche wrapper) opportuno. Oltre al segmento A/V il formato container può racchiudere anche i codec utilizzati, i sottotitoli e altri metadati. MP4, MOV, WebM e Ogg sono alcuni dei principali formati container. In questo step si sceglie anche il protocollo per lo streaming dei dati multimediali in uscita;

4. conversione, in cui il media server riceve lo stream A/V e ne effettua il trasmuxing (ossia preleva l'A/V compresso e lo reimpacchetta in un formato container differente) e/o il transcoding (cioè decomprime i contenuti). Una volta transcodificato lo stream è possibile passare alle fasi di transrating (modifica del bitrate dei contenuti decompressi) e transizing (modifica della risoluzione del video). Vengono così create diverse copie dello stream originale per adattarlo alle diverse velocità di rete e risoluzioni dei dispositivi utente;

5. distribuzione, in cui le varie versioni dello stream vengono inviate agli end user dai nodi delle Content Delivery Network (CDN), un sistema di server geograficamente distribuiti. I principali CDN commerciali a livello globale sono Akamai, Amazon Cloudfront e Cloudflare;

6. riproduzione, in cui il flusso A/V viene ricevuto dai dispositivi e infine riprodotto dagli spettatori [7].

L'architettura logica dello streaming in diretta è riassunta nello schema in figura 2.1.

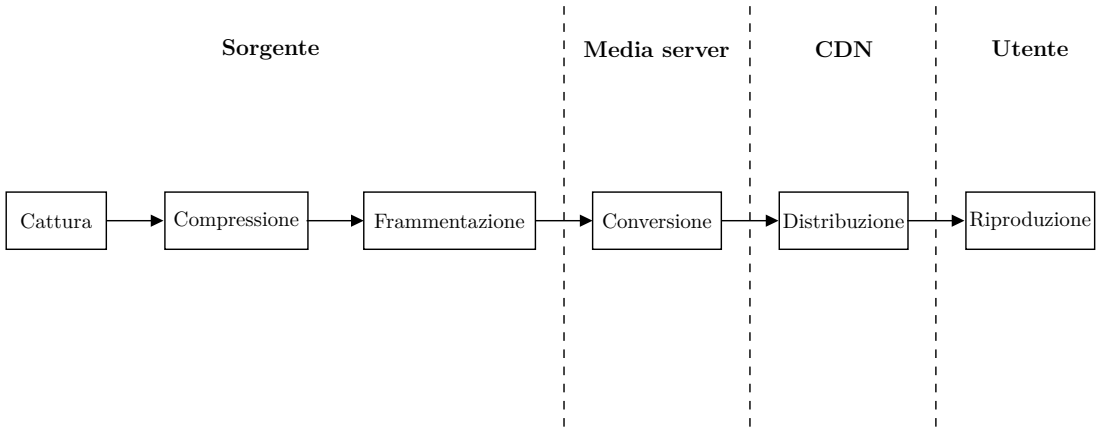

Figura 2.1: Architettura logica del processo di live streaming multimediale.

2.2 Il modello attuale

2.2.1 Implementazione del live streaming

Nello schema in figura 2.2 è riportata una possibile implementazione del processo. Vengono descritti di seguito i passaggi principali:

1. l'input A/V è il prodotto della cattura, compressione e frammentazione del flusso effettuate a bordo della sorgente;

2. il server riceve l'input, lo decodifica ed effettua la ricodifica (attraverso il media encoder) e la riframmentazione (attraverso lo stream segmenter) del flusso;

3. il contenuto multimediale è archiviato e distribuito dal web server attraverso Internet;

4. il client riceve e riproduce il contenuto multimediale.

2.2.2 Criticità del modello attuale

Tuttavia, le scelte implementative adottate nelle fasi di compressione, frammentazione e distribuzione non sono accettabili nel contesto delle LI in ambienti critici poiché causano l'insorgenza di alcune problematiche.

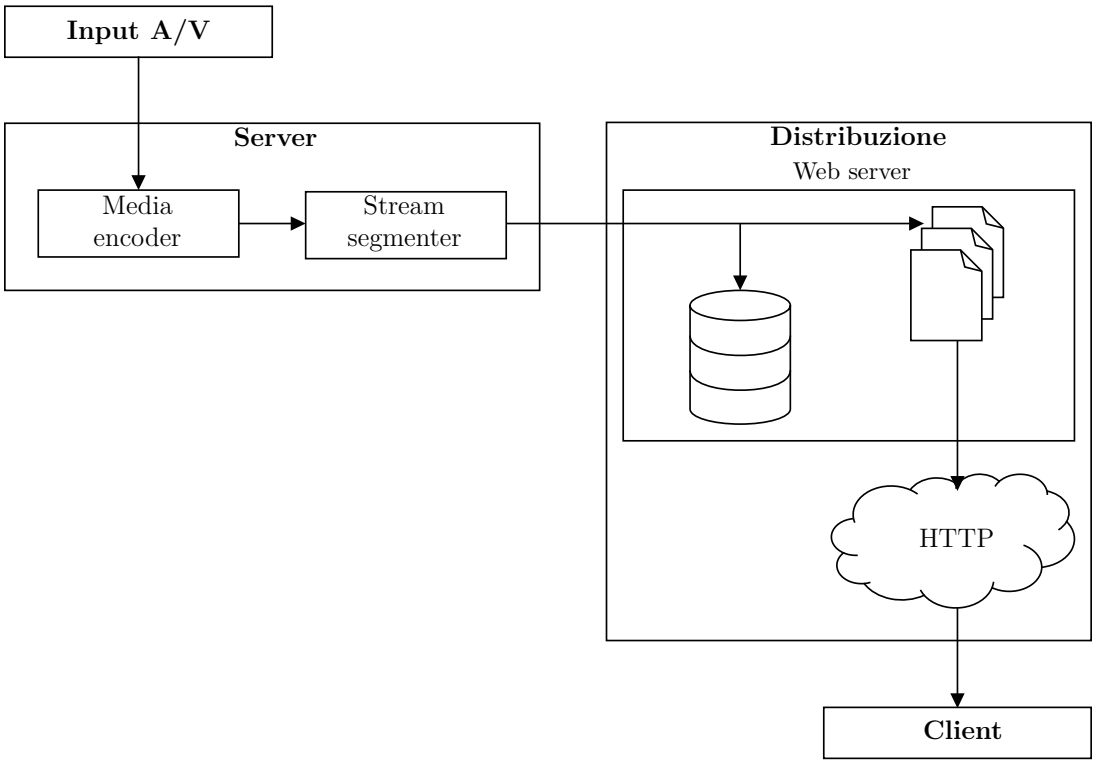

Figura 2.2: Architettura fisica del processo di live streaming multimediale.

Compressione

Per la compressione si adoperano H.264 per il video e Vorbis per l'audio. Mentre il primo funziona correttamente nel nostro contesto, il secondo è basato su un algoritmo notoriamente pesante che effettua una compressione notevole dei dati grezzi. Inoltre, per effettuare un'eventuale decodifica è necessario disporre dell'intero flusso e questo provoca ritardi inaccettabili nella catena del live streaming.

Frammentazione

I dati compressi sono frammentati e impacchettati in Ogg, un formato container basato sul Program Stream (PS) definito da MPEG. I formati basati su MPEG-PS (e di conseguenza Ogg) sono però pensati per la storicizzazione e l'archiviazione del dato e quindi sono impiegati in applicazioni multimediali affidabili, come il DVD-Video per la gestione e il trasferimento dei dati su DVD. Oltre a ciò, la dimensione eccessiva dei frammenti provoca ulteriori ritardi nell'elaborazione e nell'invio di ciascun segmento dati. Infine, solo il primo pacchetto contiene nell'intestazione le

informazioni necessarie alla codifica e decodifica del flusso: ciò significa che il client, in caso di disconnessione critica, non sa come maneggiare le informazioni all'interno dei pacchetti ricevuti successivamente al ripristino della connettività. Alla luce dei problemi esposti si può dire che Ogg non è adatto a processi come lo streaming in diretta.

Distribuzione

Per quel che concerne la distribuzione vengono impiegati protocolli tradizionali come Real-Time Streaming Protocol (RTSP) e Real-Time Messaging Protocol (RTMP). Anche se solitamente questi protocolli offrono una consegna tempestiva dei dati al client, essi non sono ottimizzati dal punto di vista dell'esperienza utente su larga scala. Inoltre, sono definiti stateful poiché per funzionare hanno bisogno di un server di streaming dedicato. Nonostante il numero di player A/V che supportano questi protocolli sia man mano diminuito nel tempo in favore di altre soluzioni, molti broadcaster scelgono RTSP o RTMP per inviare i flussi multimediali ai propri media server [7].

Strato di adattamento sul server

I problemi emersi nelle fasi di compressione e frammentazione a bordo della sorgente necessitano dell'introduzione di un workaround, rappresentato dal media encoder e dallo stream segmenter in figura 2.2, sul server: al fine di mitigare gli effetti del WiFi critico bisogna decodificare i flussi, riunirli, ricodificare e risegmentare il tutto. Questa è una soluzione decisamente inefficiente, perché si cerca di adattare alla distribuzione un flusso originariamente non adatto alla distribuzione. A causa del modo in cui la sorgente manipola i dati catturati si svolge il doppio del lavoro necessario.

2.2.3 Uno scenario esplicativo

Per presentare le criticità si passa ora in rassegna la descrizione di uno scenario abbastanza diffuso nel campo delle LI ambientali: l'intercettazione di una conversazione tra due soggetti in un autoveicolo in movimento dentro il quale è stata posizionata una microspia ambientale. Facendo riferimento al diagramma a traliccio in figura 2.3 si riportano le fasi dello scenario:

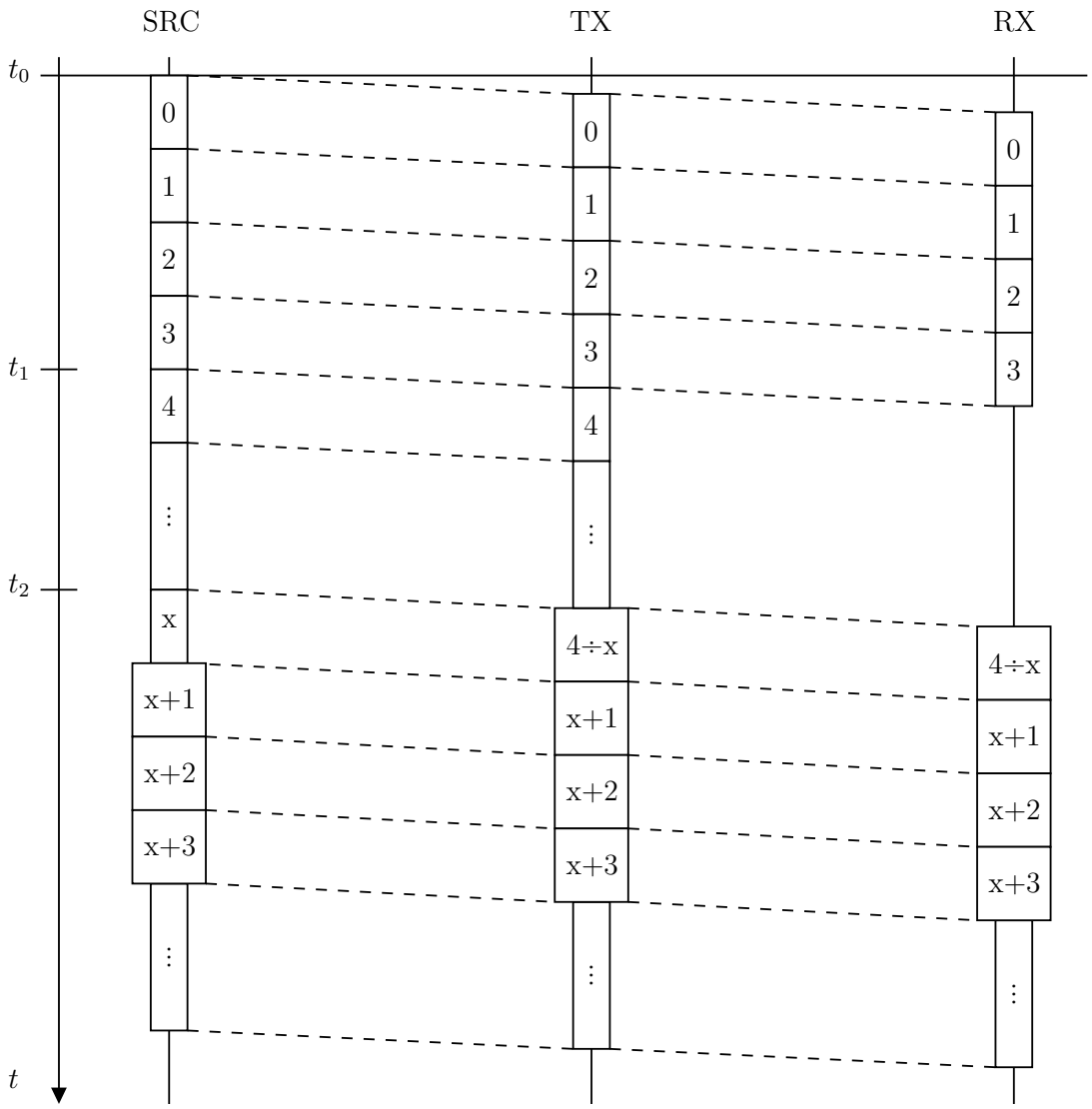

Figura 2.3: Diagramma a traliccio del caso non critico.

1. all'istante t_0 ha inizio l'intercettazione, la microspia (TX) capta la conversazione (SRC) e applica la tecnica dello store and forward, cioè salva i campioni in memoria prima di trasmetterli al ricevitore. La consegna dei dati avviene in condizioni di tempo reale o quasi;

2. all'istante t_1 l'automobile entra in una galleria e la microspia continua la registrazione dei campioni ma, non riuscendo a connettersi al ricevitore (RX), interrompe la trasmissione;

3. all'istante t_2 il mezzo esce dalla galleria e la connettività tra microspia e ricevitore viene ripristinata (si suppone che non ci siano altri fattori esterni che possano interferire tra i due). A questo punto la microspia trasmette subito i campioni registrati tra gli istanti t_1 e t_2, dopodiché ricomincia a inviare il flusso in tempo reale fino alla fine dell'intercettazione (o al sopraggiungere di una nuova interruzione);
4. al termine della LI il ricevitore ottiene un'unica registrazione dell'evento.

L'esempio appena descritto è ancora favorevole, dato che il meccanismo è resiliente a interruzioni infrequenti e di durata limitata. Tuttavia, se le disconnessioni sono più frequenti e perdurano per intervalli di tempo più lunghi si verifica quanto mostrato nel diagramma a traliccio in figura 2.4. All'istante t_2 la connettività tra microspia e ricevitore è ripristinata, ma i campioni registrati tra t_1 e t_2 e salvati a bordo della microspia sono inviati solo in un secondo momento attraverso un canale di comunicazione diverso da quello in uso. Inoltre, la microspia interrompe la registrazione iniziata all'istante t_0 e ne inizializza una nuova.

È bene rendere ancora più evidente che la principale debolezza di questo approccio non è rappresentata dalla perdita di contenuti legali: tale circostanza, infatti, non è contemplata. Il punto critico del protocollo e dei formati adottati consiste invece nella frammentazione delle conversazioni multimediali intercettate: non è possibile alcun meccanismo diretto per evitare tali situazioni senza dover ricorrere ad alcuni workaround che includano tecniche di post-processing sui flussi ricevuti al fine di ricomporre le conversazioni intercettate.

2.3 Verso la definizione di un nuovo modello

A questo punto, urge la necessità di sostituire il processo attuale a favore di uno più resiliente e adatto alla trasmissione di flussi multimediali, che devono essere segmentati e formattati opportunamente: in questo modo, in caso di connettività scarsa o assente, è possibile prevenire eventuali frammentazioni dei dati garantendo così la continuità del flusso multimediale e, di conseguenza, una riproduzione coerente dei contenuti.

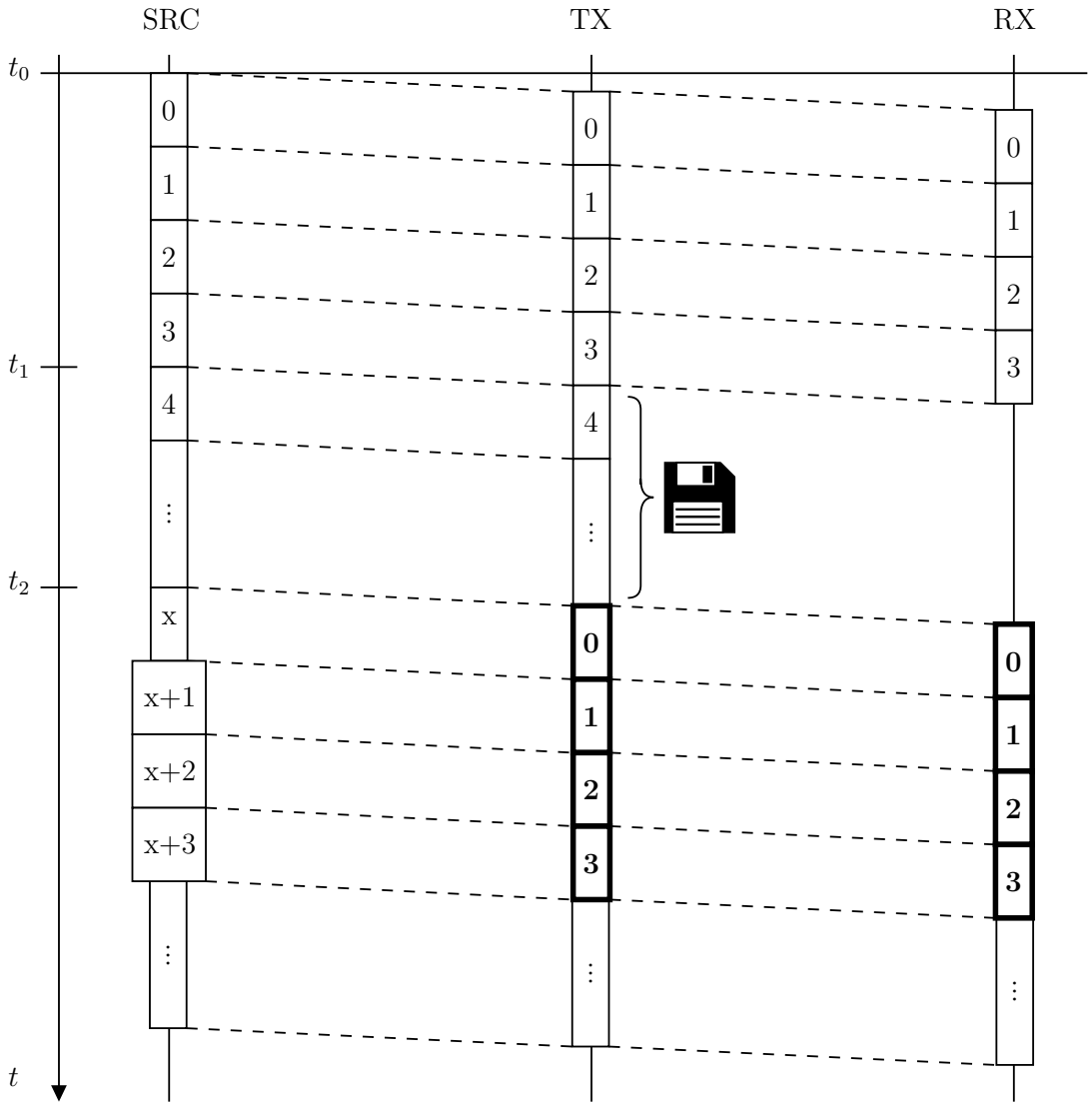

Figura 2.4: Diagramma a traliccio del caso critico.

Capitolo 3

Soluzione proposta

3.1 Il nuovo modello

La nuova implementazione del processo di live streaming è riportata in figura 3.1. Rispetto al modello attuale presentato nel paragrafo 2.2, sono state apportate le seguenti migliorie:

- la compressione dell'audio viene effettuata con Opus, più leggero di Vorbis in quanto effettua una compressione minore garantendo comunque una qualità più alta. In aggiunta, Opus non ha bisogno di ricevere l'intero flusso per effettuarne la decodifica. Per quanto riguarda il video, H.264 garantisce prestazioni più che accettabili, ma si può optare per H.265, più efficiente ma più dispendioso di potenza di calcolo;
- per la frammentazione si ricorre al formato container MPEG-TS, presentato nel paragrafo 3.2;
- la distribuzione è regolata dal protocollo HTTP-based adattivo HLS, esaminato nel paragrafo 3.3.

A questo punto il server non ha più bisogno dello strato di adattamento formato da media encoder e stream segmenter: infatti, esso si limita semplicemente ad estrarre gli header per le operazioni di signalling e concatenare i payload preservando la qualità del flusso.

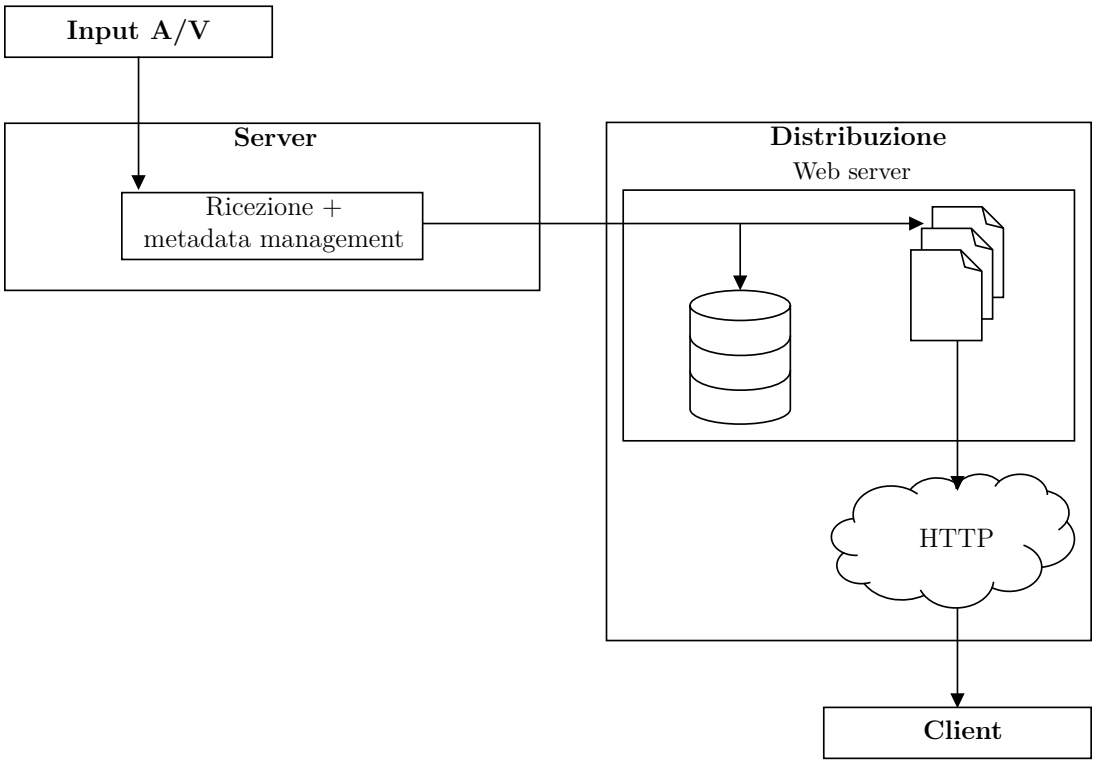

Figura 3.1: La nuova architettura fisica del processo di live streaming.

3.2 Il formato MPEG-TS

3.2.1 Definizione

L'MPEG-2 Transport Stream è un formato, definito dal comitato Moving Picture Experts Group (MPEG) nello standard ISO/IEC 13818-1 [8], utilizzato come contenitore di flussi di pacchetti Packetized Elementary Streams (PES). Un pacchetto PES è il risultato della codifica, della compressione e della pacchettizzazione di flussi audio, video o di dati generici (si veda il diagramma in figura 3.2).

3.2.2 Applicazioni

Il Transport Stream è stato originariamente impiegato nella diffusione dei flussi audiovisivi in broadcast per la televisione digitale terrestre e via cavo. Con il passare del tempo, lo standard è stato adottato sempre più nello streaming satellitare e

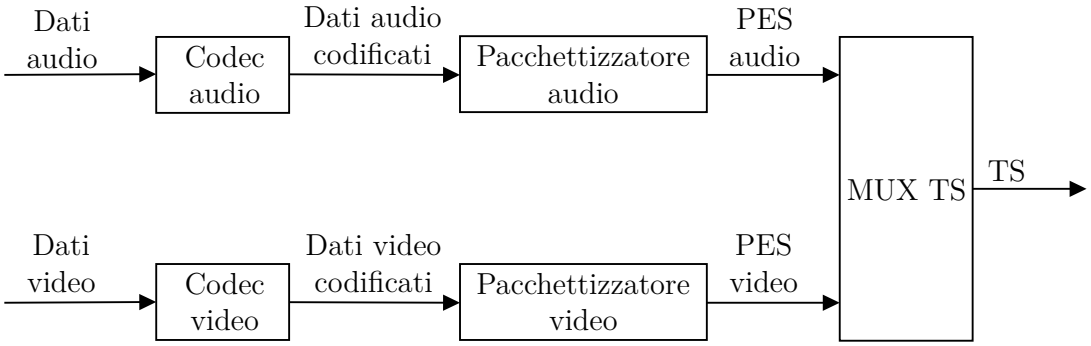

Figura 3.2: Diagramma del processo di creazione del TS.

su reti a lunga distanza. Queste applicazioni, poiché i dati compressi vengono trasmessi in tempo reale attraverso protocolli con minore affidabilità, come ad esempio UDP, sono spesso soggette a perdite di pacchetti ed errori sui valori di bit.

3.2.3 Vantaggi

Ogni TS è suddiviso in pacchetti, interlacciati tra loro, di dimensione massima 188 byte. Generalmente, i primi 4 byte servono per l'header, ed i restanti 184 sono lasciati al payload. Questa dimensione è un valido compromesso tra:

- il numero di pacchetti, poiché ognuno di essi deve essere generato ed inviato al destinatario. Inoltre, per ogni pacchetto è necessario instanziare il codificatore, codificare il pacchetto e deistanziare il codificatore. Questa sequenza di operazioni, se ripetuta per un numero eccessivo di volte, ha un effetto negativo sulle prestazioni del sistema;

- la latenza, che cresce al crescere della dimensione del pacchetto.

A differenza degli altri formati contenitori di uso comune come AVI, MOV/MP4 e MKV, la dimensione ridotta dei pacchetti consente di trasmettere i dati con una minore latenza e con una maggiore capacità di recupero dopo un errore [9]. Questo è un aspetto particolarmente rilevante nelle LI in ambienti WiFi critici, poiché la scelta di impiegare pacchetti di grandi dimensioni potrebbe comportare dei ritardi inaccettabili. Infine, mentre in Ogg solo il primo pacchetto possiede nell'header le informazioni necessarie per manipolare il flusso, in MPEG-TS tutti i pacchetti

le racchiudono: di conseguenza, MPEG-TS rende la trasmissione resiliente alle disconnessioni di rete che si verificano nei contesti a connettività limitata.

3.3 Il protocollo HLS

3.3.1 Definizione

HTTP Live Streaming (HLS) è un protocollo di comunicazione per lo streaming a bitrate adattivo (adaptive bitrate streaming) di contenuti multimediali in rete sia in diretta che on demand.

Sviluppato da Apple e rilasciato nel 2009, HLS fu inizialmente implementato come parte integrante di QuickTime, Safari e di alcune applicazioni presenti esclusivamente sui dispositivi macOS e iOS. In poco tempo, il supporto in un numero sempre più grande di applicazioni, tra cui riproduttori multimediali, web browser e dispositivi mobile (come dimostra la tabella 3.1), ha reso HLS lo standard de facto per lo streaming a bitrate adattivo [10].

3.3.2 Funzionamento

Il funzionamento del protocollo, descritto dal RFC 8216 [12], è abbastanza semplice: l'intero flusso del contenuto da trasmettere (ad esempio un filmato) viene scomposto in file più piccoli, detti chunk, media segment o segmenti, che presentano le seguenti caratteristiche:

- sono di durata generalmente compresa tra 2 e 10 secondi;
- sono nel formato TS;
- incapsulano audio, video o entrambi;
- possono essere scaricati separatamente alla stregua di un tradizionale download di file via HTTP.

Inoltre, ciascuno dei chunk che compongono il flusso viene generato in più versioni (definite variant stream) con diverse risoluzioni, bitrate e frequenza di fotogrammi

Client	**Piattaforma**	**Editor**	**Supporto a HLS a partire dalla versione**
Safari	macOS, iOS	Apple	6.0
Google Chrome / Chromium	Windows, macOS, Linux, Android, iOS	Google	30
Firefox	Windows, macOS, Linux, Android, iOS	Mozilla	50.0 (Android) 57.0 (altre piattaforme)
Microsoft Edge	Windows 10	Microsoft	EdgeHTML 12
QuickTime Player	macOS	Apple	10.0
iTunes	Windows, macOS	Apple	10.1
VLC media player	Windows, macOS, Linux, Android, iOS, Windows Phone	VideoLAN	3.0
QuickPlayer	Windows 7/8/8.1/10, Android, iOS	Squadeo	-

Tabella 3.1: Elenco dei principali client che supportano HLS [11].

chiave (keyframe), affinché durante la riproduzione l'utente possa selezionare dinamicamente un livello differente di qualità del flusso, passando al download del segmento nella versione specifica, o consentendo al player di scegliere automaticamente il formato più adatto alla velocità ed allo stato della rete (figura 3.3).

I contenuti audio trasmessi con HLS sono codificati principalmente in AAC e MP3, mentre quelli video in H.264.

Il flusso è descritto tramite una playlist (detta anche manifest), un file testuale con codifica UTF-8 e in formato m3u8, che può essere:

- una master playlist, se contiene i riferimenti ad altre playlist che raggruppano i segmenti dei video per una determinata versione del flusso (come nello schema in figura 3.4);
- una media playlist, se contiene i metadati del flusso, cioè i dettagli dei chunk che lo compongono.

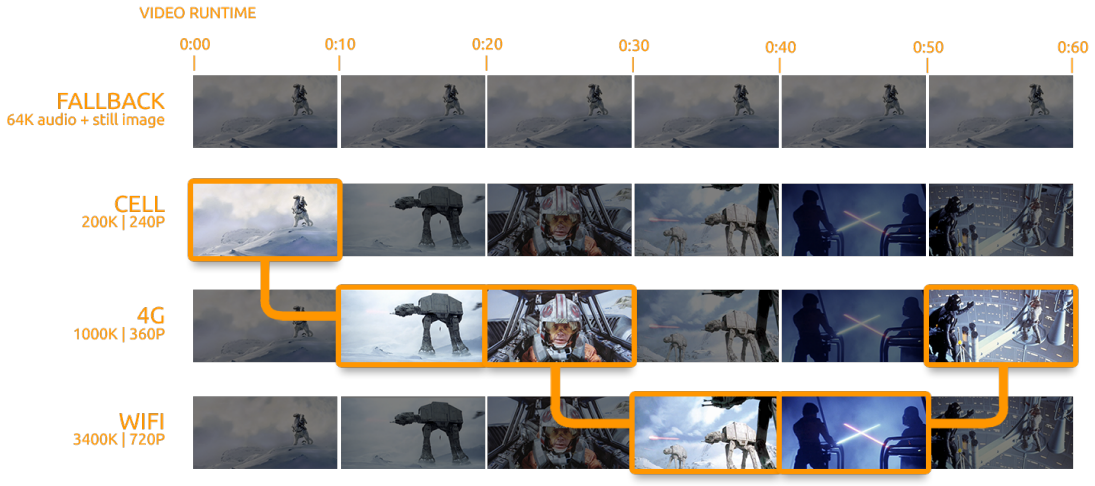

Figura 3.3: Processo di scelta dei chunk a diversa risoluzione [13].

La playlist funge da entry point: bisogna specificare la sua Unified Resource Identifier (URI) per riprodurne il contenuto.

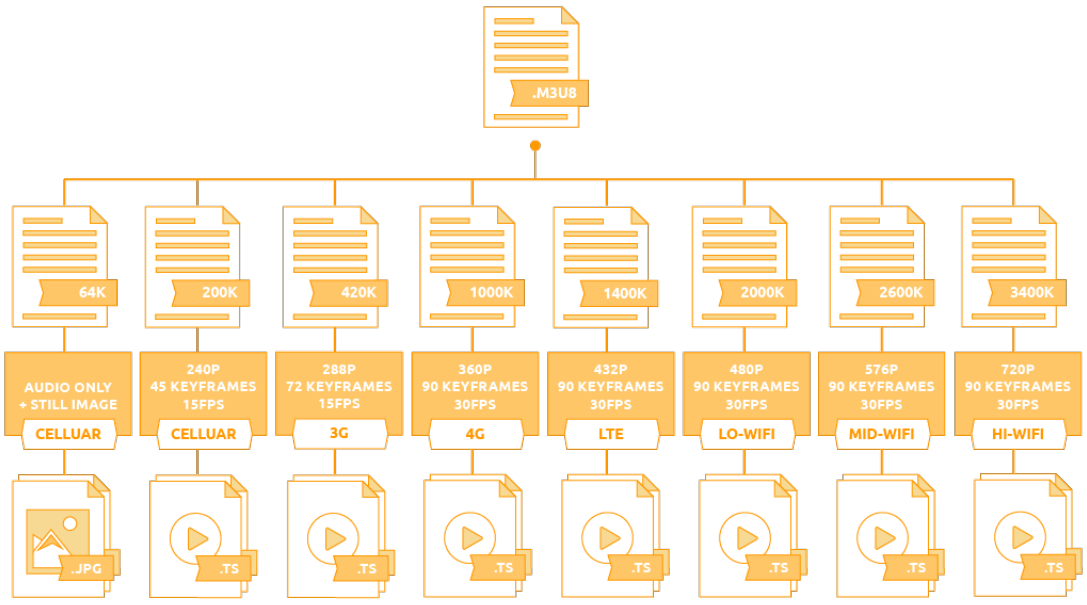

Figura 3.4: Struttura di una master playlist (in alto) composta da più media playlist (al centro) [13].

Master playlist

Un esempio di master playlist è riportato nel listato 3.1. Le keyword precedute da #, dette metatag, sono descritte di seguito:

```
1 #EXTM3U
2 #EXT-X-STREAM-INF:BANDWIDTH=1280000,CODECS="mp4a.40.2,avc1.4d401e"
3 http://example.com/low_res.m3u8
4 #EXT-X-STREAM-INF:BANDWIDTH=2560000,CODECS="mp4a.40.2,avc1.4d401e"
5 http://example.com/mid_res.m3u8
6 #EXT-X-STREAM-INF:BANDWIDTH=7680000,CODECS="mp4a.40.2,avc1.4d401e"
7 http://example.com/hi_res.m3u8
8 #EXT-X-STREAM-INF:BANDWIDTH=65000,CODECS="mp4a.40.5"
9 http://example.com/audio_only.m3u8
```

Listato 3.1: Esempio di una master playlist minima.

- `EXTM3U` indica che la playlist è un file nel formato M3U esteso. Tutte le playlist HLS devono presentare questo metatag all'inizio del file;
- `EXT-X-STREAM-INF` segnala che l'URI riportata nella riga successiva rappresenta una media playlist. Si presenta nel formato:

  ```
  #EXT-X-STREAM-INF:<attribute-list>
  <URI>
  ```

  dove:

  - `attribute-list` è un elenco di attributi separati da virgola. Nel listato 3.1 sono stati riportati i seguenti attributi:
    - `BANDWIDTH`, obbligatorio, è un intero che indica il limite superiore del bitrate (in frame/s) del variant stream;
    - `CODECS`, fortemente raccomandato, è una stringa contenente una lista di formati, separati da virgola, ciascuno dei quali specifica un media type presente in un frammento della playlist. I formati validi sono specificati nel RFC 6381 [14];
  - `URI` specifica una media playlist. Nel listato 3.1, le media playlist sono:

    * http://example.com/low_res.m3u8;
    * http://example.com/mid_res.m3u8;
    * http://example.com/hi_res.m3u8.

  Quindi, low_res.m3u8 ha un bitrate di 1280 kbps, e contiene:

  – audio nel formato AAC Low Complexity (AAC-LC);
  – video nel formato H.264 Main Profile Level 3.

Media playlist

In maniera analoga alla master playlist si riporta un esempio di media playlist nel listato 3.2 e si descrivono i metatag presenti al suo interno:

```
1  #EXTM3U
2  #EXT-X-TARGETDURATION:10
3  #EXT-X-VERSION:3
4  #EXTINF:10.000,
5  http://media.example.com/segment_01.ts
6  #EXTINF:10.000,
7  http://media.example.com/segment_02.ts
8  #EXTINF:5.000,
9  http://media.example.com/segment_03.ts
10 #EXT-X-ENDLIST
```

Listato 3.2: Esempio di una media playlist minima.

- EXTM3U, come nella master playlist, definisce una playlist M3U;
- EXT-X-TARGETDURATION, obbligatorio, definisce la Target Duration, un limite superiore della durata dei chunk nella playlist. Si presenta nel formato:

  #EXT-X-TARGETDURATION:<s>

  dove s è un numero intero o un decimale che indica la Target Duration in secondi;
- EXT-X-VERSION indica la versione da adottare per la compatibilità tra la playlist, i suoi media ed il server. Il metatag viene applicato all'intera playlist.

Un file di playlist non può contenere più di un metatag `EXT-X-VERSION`. Si presenta nel formato:

```
#EXT-X-VERSION:<n>
```

dove `n` è un numero intero che indica il numero di versione del protocollo;

- `EXTINF` indica la durata del media segment indicato. Il metatag è obbligatorio per ogni media segment. Si presenta nel formato:

  ```
  #EXTINF:<duration>,[<title>]
  ```

  dove:

  – `duration` è un numero intero o decimale che indica la durata del media segment;

  – `title`, opzionale, è il titolo, in formato UTF-8, del media segment;

- `EXT-X-ENDLIST` indica che non ci sono più media segment da aggiungere alla playlist.

3.3.3 Vantaggi

HLS e i protocolli HTTP-based adattivi

I protocolli basati su HTTP sono adattivi, ossia regolano la qualità del flusso in base allo stato della rete e in questo modo garantiscono sempre la migliore esperienza utente possibile a prescindere dalla connettività, dal software o dai dispositivi in uso. A differenza dei protocolli tradizionali, essi sono stateless e sono quindi compatibili anche con web server generici [7].

Flessibilità del protocollo

HLS rappresenta la principale svolta nel campo delle intercettazioni in ambienti critici: se si verifica una disconnettività di rete, esso è in grado di consegnare i pacchetti "in disordine", alternando cioè tempo reale e differita. Il protocollo, dunque, ignora le lacune nella trasmissione e il ricevitore inserisce i segmenti della registrazione al posto giusto quando li riceve.

Supporto per tutti i dispositivi

Sono ormai pochi i dispositivi privi di supporto a HLS, ed è proprio questo uno dei motivi fondamentali per cui il protocollo oggi è considerato uno degli standard più popolari ed apprezzati nel campo dello streaming a bitrate adattivo (figura 3.5). Inoltre, anche i principali web server e i CDN come Akamai riescono a gestire le trasmissioni con HLS in maniera efficiente [15].

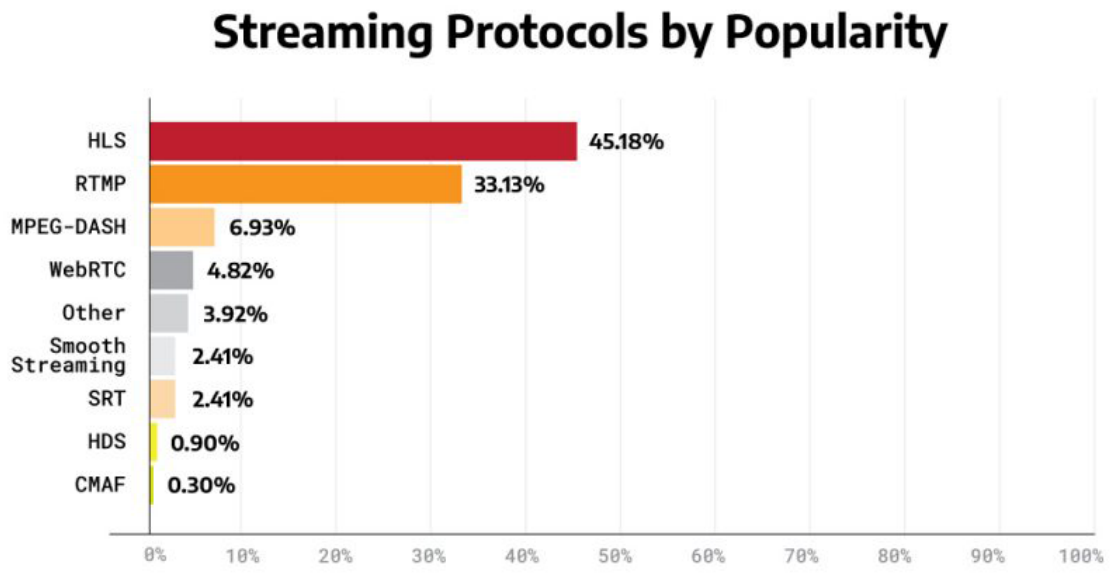

Figura 3.5: Istogramma della popolarità dei principali protocolli di streaming a settembre 2020 [16].

Esperienza d'uso eccellente

Grazie allo streaming a bitrate adattivo, HLS, se configurato in maniera opportuna, cerca di trasmettere il video alla migliore qualità consentita della rete, minimizzando così il ritardo ed il buffering [15].

Privacy e sicurezza

È possibile utilizzare la crittografia per proteggere i flussi multimediali trasmessi su HLS. Uno degli algoritmi più utilizzati è l'AES-128 [17].

Supporto ai file MP4

Durante la WorldWide Developers Conference (WWDC) del 2016 Apple ha introdotto la possibilità di trasmettere anche file MP4 opportunamente frammentati, alla stregua dei frammenti TS. Con questo passo in avanti, la distanza tra HLS ed il protocollo MPEG-DASH è stata accorciata e la compatibilità tra i due aumentata [18].

Gestione della latenza

Come tutti i protocolli, HLS, nonostante i numerosi pregi, non è esente da difetti: uno dei più importanti è la latenza. In pratica, la latenza è l'intervallo di tempo t che intercorre tra l'istante t_1 in cui un evento si verifica e l'istante t_2 in cui l'utente lo visualizza sul proprio dispositivo. Durante l'intervallo di tempo t, il video dell'evento viene registrato, processato dal codificatore, trasmesso sulla rete, distribuito e decodificato dal client affinché l'utente possa visualizzarlo.

Come mostrato in figura 3.6, adoperando il protocollo HLS con segmenti di durata pari o superiore a 10 s si ottiene una latenza maggiore di 30 s. Come descritto nel sottoparagrafo 1.2.2, un simile valore per la latenza non è accettabile nel contesto delle LI in ambienti WiFi critici.

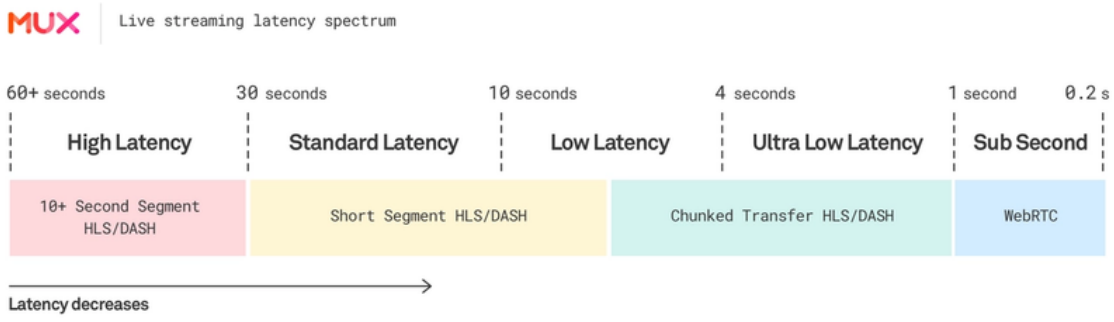

Figura 3.6: Definizione degli intervalli di latenza e dei relativi protocolli [19].

In genere, è possibile ridurre la latenza riducendo la durata dei chunk del video; questo, però, comporta un incremento del numero di richieste HTTP, ognuna delle quali richiede l'invio di un overhead preliminare di dimensioni non trascurabili [19].

Ma la soluzione definitiva è arrivata proprio da Apple: durante la WWDC del 2019, la multinazionale di Cupertino ha presentato Low-Latency HLS (LL-HLS), un'estensione basata sul protocollo esistente, quasi una sorta di HLS 2.0, che consente di trasmettere flussi multimediali in rete garantendo latenze non superiori ai 2 s. Inoltre, LL-HLS è compatibile all'indietro: i player che non lo supportano possono ripiegare sull'HLS standard [20]. Un ulteriore passo in avanti è stato compiuto nel 2020, quando Apple, grazie al prezioso contributo di numerosi fornitori (tra cui Akamai, Wowza e AWS Elemental) [19], ha reso LL-HLS parte integrante dello standard HLS [21].

Resilienza e resistenza ad estensioni da parte dell'utente

Si consideri una media playlist funzionante, a cui sono aggiunti alcuni metatag definiti dall'utente (riportati in grassetto nel listato 3.3). Riproducendo il contenuto della playlist, si constata l'assenza di errori: HLS è robusto, e ignora queste estensioni impreviste. Inoltre, l'utente può estendere il set di metatag aggiungendone di propri, in base alle proprie esigenze, e modificando opportunamente il codice di HLS in modo che il protocollo riconosca queste estensioni.

```
1  #EXTM3U
2  #EXT-X-TARGETDURATION:10
3  #MY-METATAG-1
4  #EXT-X-VERSION:3
5  #EXTINF:10.000,
6  #MY-METATAG-2
7  http://media.example.com/segment_01.ts
8  #EXTINF:10.000,
9  http://media.example.com/segment_02.ts
10 #MY-METATAG-3
11 #EXTINF:5.000,
12 http://media.example.com/segment_03.ts
13 #EXT-X-ENDLIST
```

Listato 3.3: Media playlist con metatag definiti dall'utente.

In questo modo, si crea in HLS un canale dedicato al trasferimento di dati, simile a quelli relativi ai flussi audiovisivi (come mostrato in figura 3.7), con i quali condivide

la medesima base dei tempi.

Figura 3.7: Astrazione del nuovo canale dati in HLS.

L'utente può dichiarare informazioni aggiuntive, inserirle nella playlist e trasmetterle al ricevitore alla stregua dei flussi A/V.

Awareness del player

Alla luce delle modifiche sopracitate e della loro applicazione nel contesto presentato nel capitolo 1, si può parlare di awareness (consapevolezza) del player: esso, grazie alle estensioni definite dall'utente e ottenute dal ricevitore attraverso il canale dati di HLS, è a conoscenza sia dello stato del trasmettitore, cioè il dispositivo di intercettazione nel contesto delle LI ambientali, che della rete su cui viaggiano i dati. I dati trasmessi possono contenere informazioni molto utili ai fini delle LI, tra cui:

- informazioni sullo stato del trasmettitore come il tempo attuale, le coordinate GPS, la temperatura, il voltaggio, la batteria residua etc.;
- informazioni sullo stato della rete come la velocità di trasferimento e la capacità del canale di trasmissione.

3.4 Dalla teoria alla pratica

Sulla carta, il connubio tra il formato contenitore MPEG-TS e il protocollo HLS sembra risolvere (o, laddove non sia possibile, mitigare) le criticità che la soluzione attuale introduce nel processo di live streaming, e in particolare in quello delle LI.

Adesso si rende necessario l'avvio di una fase di sperimentazione preliminare per appurare o confutare la validità della soluzione proposta.

Capitolo 4

Sviluppo

4.1 Analisi di fattibilità

Gli obiettivi prefissati per l'analisi di fattibilità sono i seguenti:

1. creare un player A/V di base per la riproduzione di un filmato locale;
2. implementare il protocollo HLS nel player attraverso una libreria dedicata;
3. estrarre le tracce audio e video dai segmenti della playlist;
4. indirizzare le tracce video verso l'elemento `<video>` esistente;
5. indirizzare le tracce audio verso un nuovo elemento `<audio>`;
6. implementare il canale dati tramite l'aggiunta e l'intercettazione di metatag custom.

4.1.1 Creazione del player

Lo step iniziale di questa fase consiste nella creazione di una semplice pagina HTML (`index.html`, listato 4.1) in cui sono definiti un elemento `<video>` e i `<button>` per le operazioni basilari di controllo del video (play, pausa, stop, indietro e avanti).

```
1 <!DOCTYPE html>
2 <html>
```

```
3   <head>
4     <meta charset="UTF-8">
5     <title>Test</title>
6     <script src="myScript.js" defer></script>
7   </head>
8   <body>
9
10    <!-- Elemento video -->
11    <video id="video">
12      <source src="myVideo.mp4" type="video/mp4">
13    </video>
14
15    <!-- Controlli -->
16    <button type="button" id="play">Play</button>
17    <button type="button" id="pause">Pausa</button>
18    <button type="button" id="stop">Stop</button>
19    <button type="button" id="bwd">Indietro 5s</button>
20    <button type="button" id="fwd">Avanti 5s</button>
21
22  </body>
23 </html>
```

Listato 4.1: `index.html` - Pagina HTML minimale per il corretto funzionamento del player di base.

Successivamente si definisce il codice JavaScript (`myScript.js`, listato 4.2) per la gestione del video al click dei pulsanti.

```
1  var video = document.getElementById("video");
2
3  var controls = {
4      play: document.getElementById("play"),
5      pause: document.getElementById("pause"),
6      stop: document.getElementById("stop"),
7      backward: document.getElementById("bwd"),
8      forward: document.getElementById("fwd")
9  }
10
11 controls.play.addEventListener("click", function(){
```

```
12      video.play();
13  });
14  controls.pause.addEventListener("click", function(){
15      video.pause();
16  });
17  controls.stop.addEventListener("click", function(){
18      video.currentTime = 0;
19      video.pause();
20  });
21  controls.bwd.addEventListener("click", function(){
22      video.currentTime -= 5;
23  });
24  controls.fwd.addEventListener("click", function(){
25      video.currentTime += 5;
26  });
```

Listato 4.2: `myScript.js` - Codice JavaScript minimale per il corretto funzionamento del player di base.

Dopo una serie di modifiche allo stile della pagina e dei suoi elementi tramite CSS si ottiene il player mostrato in figura 4.1.

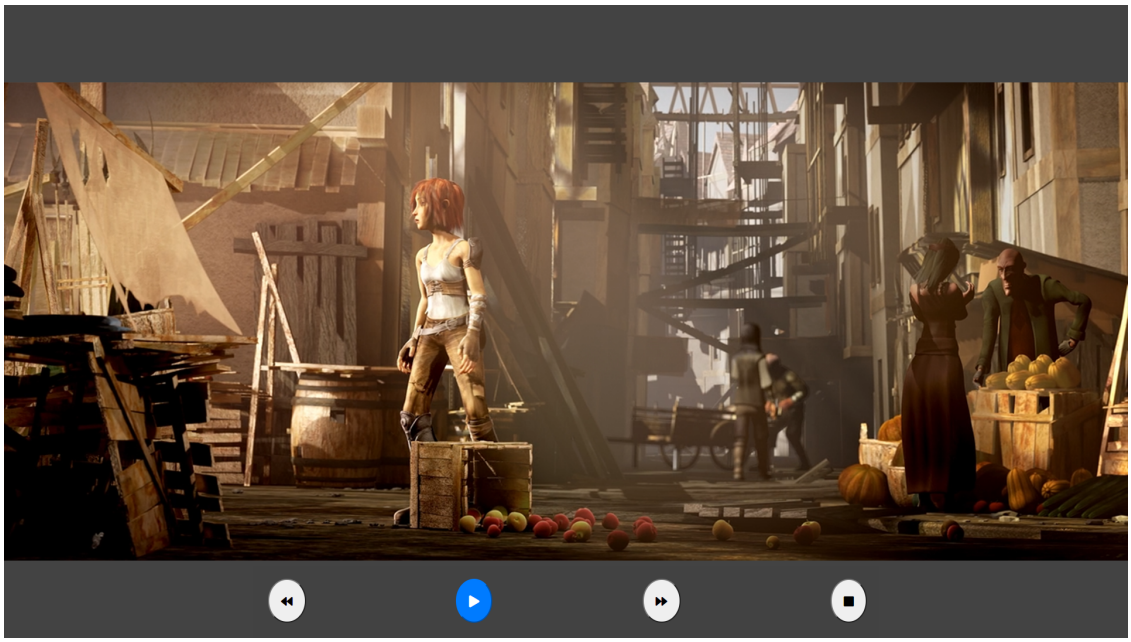

Figura 4.1: Il player A/V di base.

Il prossimo passo è la scelta della libreria HLS da implementare. Prima di proseguire, è necessario fare una digressione sulle librerie disponibili in rete per implementare il protocollo HLS sul proprio browser.

4.1.2 Panoramica delle librerie HLS

Quando Apple rilasciò HLS per i propri dispositivi, in Safari era possibile semplicemente aggiungere la seguente riga:

```
<video src='path/to/playlist.m3u8'></video>
```

nel body di una pagina HTML per ottenere un player video funzionante. Per tali strumenti, il player web integrato è già compatibile con il protocollo HLS.

I browser Chrome e Firefox implementano la tecnologia Web Audio/Video ma, per ragioni difficilmente interpretabili, non integrano HLS: di conseguenza, i dispositivi sono stati riprogettati per consentire lo streaming nella modalità indicata dal browser in uso.

Librerie Flash

Col passare degli anni, sono state rilasciate diverse librerie in Adobe Flash, come Flashls e osmfhls nel 2014, per il supporto a HLS per i browser desktop. Tuttavia, Flash era una tecnologia notoriamente lenta, vulnerabile, con requisiti hardware esosi e livelli di usabilità e accessibilità mediocri. Già in quegli anni tra gli esperti si vociferava della morte della criticata piattaforma in un futuro non molto remoto: effettivamente, Flash fu abbandonato nel 2020 a favore di altre piattaforme già esistenti come HTML5 e WebGL. Erano dunque necessarie delle alternative più performanti e affidabili.

Librerie JavaScript

Per far fronte alle negligenze delle librerie Flash sono nate alcune librerie JavaScript, come hls.js, video.js, hasplayer.js e Shaka Player. Queste librerie sono:

- open-source;
- scritte nel linguaggio JavaScript;

- costruite attorno agli elementi HTML5 `<video>` e `<audio>` e si basano sulle Media Source Extensions (MSE) e le Encrypted Media Extensions (EME), un insieme di API del World Wide Web Consortium (W3C) che fanno da collante tra HTML5 e HLS;
- compatibili con la maggior parte dei browser attuali, sia desktop che mobile, purché questi supportino le MSE;
- molto più performanti delle librerie Flash [22].

4.1.3 La libreria hls.js

Per questo lavoro, è stata scelta hls.js [23]. Questa libreria è stata adottata da un numero sempre più grande di piattaforme, tra cui Twitter, DailyMotion e Akamai, che la impiegano in fase di production.

Architettura

Il codice di hls.js è scritto sia in JavaScript che in TypeScript, un linguaggio open source basato su ECMAScript 6 e che estende JavaScript. Esso viene dunque transpilato in ECMAScript 5 tramite il compilatore di TypeScript e Babel. Infine, Webpack genera il bundle da fornire al browser.

La libreria hls.js è divisa in moduli, o sottosistemi, ciascuno dei quali assolve una o più funzioni. Ad ogni modulo corrisponde uno o più file sorgente e i moduli possono essere organizzati nei gruppi mostrati nell'architettura in figura 4.2:

- HLS Core definisce la classe HLS e istanzia i moduli necessari al suo funzionamento;
- Controller raggruppa i moduli con funzione di monitoraggio dei vari component (tra cui i buffer, lo stream etc.) della libreria;
- Loader comprende i moduli impegnati nelle interazioni tra la libreria e la rete;
- Utilities rappresenta un gruppo eterogeneo di moduli, tra cui il parser della playlist M3U8 e il logger per le stampe di debug;
- Data structures racchiude le strutture dati utilizzate all'interno di hls.js.

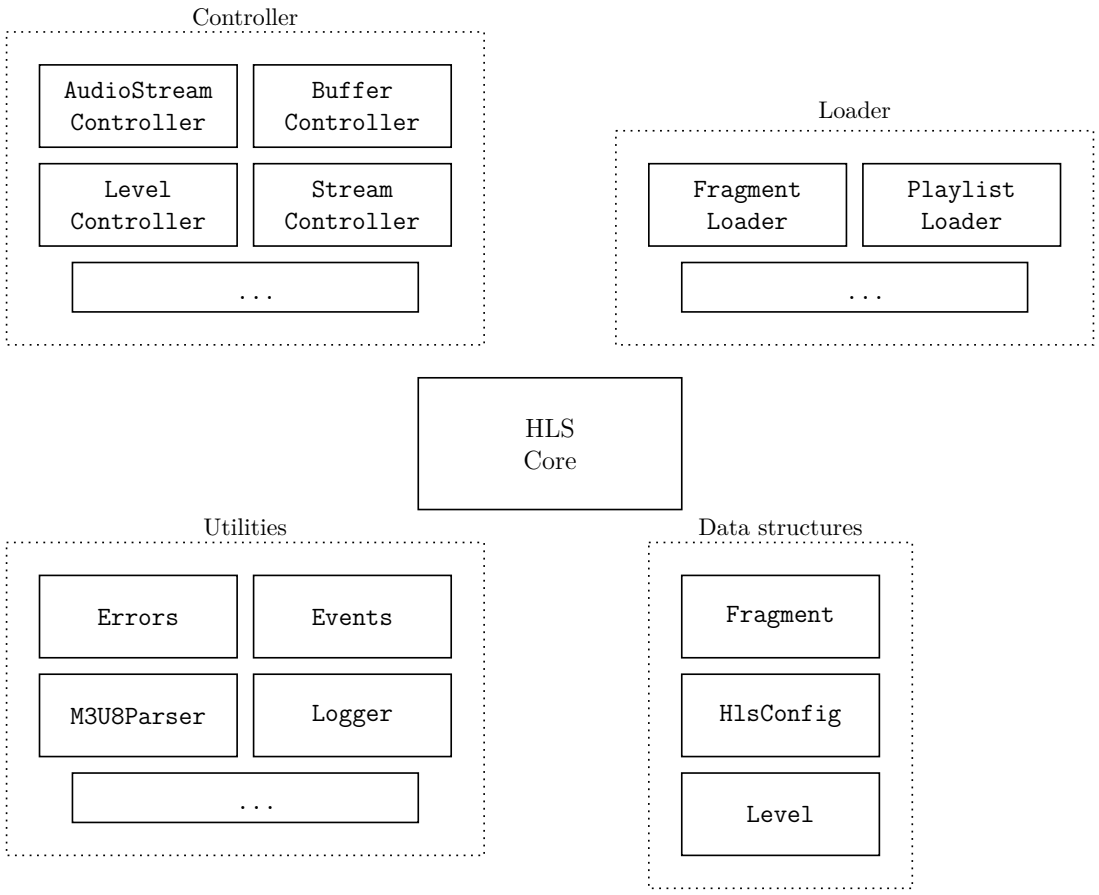

Figura 4.2: Architettura di hls.js.

Le comunicazioni, sia all'interno dello stesso modulo che tra due o più moduli, sono regolate da un sistema basato su eventi gestito dall'`EventEmitter`. Il listato 4.3 mostra la definizione in `events.js` dell'oggetto `HlsEvents`, una `struct` i cui campi rappresentano i singoli eventi della libreria, mentre il listato 4.4 illustra il meccanismo di innesco e intercettazione degli eventi [24].

```
1 const HlsEvents = {
2   // Scatenato prima dell'aggancio di un MediaSource a un media element
3   MEDIA_ATTACHING: 'hlsMediaAttaching',
4   // Scatenato dopo l'aggancio di un MediaSource a un media element
5   MEDIA_ATTACHED: 'hlsMediaAttached',
6   ...
7 };
8 export default HlsEvents;
```

Listato 4.3: Definizione degli eventi in hls.js.

```
import Event from 'events';
import EventHandler from 'event-handler';

class SampleModule extends EventHandler {
 constructor(hls) {
 super(hls,
  /* Elenco degli eventi che il modulo deve intercettare */
  Event.EVENT_NAME,
  ...);
 }
 /* Innesco di EVENT_NAME */
 this.hls.trigger(Event.EVENT_NAME, {
  param1: value1,
  param2: value2,
  ...
 });
 /* Intercettazione di EVENT_NAME */
 this.onEventName(data){
  console.log(data.param1);
  console.log(data.param2);
  ...
 }
}
```

Listato 4.4: Meccanismo di gestione degli eventi in hls.js.

4.1.4 Configurazione di hls.js

Si riprenda ora la pagina HTML creata nel listato 4.1 e il codice JavaScript del listato 4.2. Seguendo le indicazioni riportate nel file `README` della repository di hls.js [23], si incorpora la libreria (listato 4.5).

```
<!DOCTYPE html>
<html>
  <head>
```

```
4       ...
5       <script src="https://cdn.jsdelivr.net/npm/hls.js"></script>
6     </head>
7     ...
8   </html>
```

Listato 4.5: `index.html` - Incorporazione di hls.js in una pagina HTML.

Facendo riferimento al listato 4.6 si riportano i passaggi principali per il corretto funzionamento di hls.js:

1. si istanzia un oggetto HLS (riga 6);
2. si aggancia il `<video>` all'oggetto HLS (riga 7);
3. si avvia il caricamento della playlist (riga 11);
4. si definiscono le operazioni da effettuare in caso di errori (righe 18÷35).

```
1   ...
2
3   var videoSrc = 'http://my.streamURL.com/playlist.m3u8';
4
5   if (Hls.isSupported()) {
6    var hls = new Hls();
7    hls.attachMedia(video);
8    // L'evento MEDIA_ATTACHED è scatenato appena il MediaSource è pronto
9    hls.on(Hls.Events.MEDIA_ATTACHED, function () {
10    console.log('Video agganciato a hls.js con successo.');
11    hls.loadSource(videoSrc);
12    hls.on(Hls.Events.MANIFEST_PARSED, function (event, data) {
13     console.log('Manifest caricato con successo.');
14    });
15   });
16
17   /* Gestione degli errori */
18   hls.on(Hls.Events.ERROR, function (event, data) {
19    if (data.fatal) {
20     switch (data.type) {
21      case Hls.ErrorTypes.NETWORK_ERROR:
```

```
22        console.log('Errore di rete. Ripristino in corso...');
23        hls.startLoad();
24        break;
25       case Hls.ErrorTypes.MEDIA_ERROR:
26        console.log('Errore sul media. Ripristino in corso...');
27        hls.recoverMediaError();
28        break;
29       default:
30        console.log('Errore non gestito. Distruzione di HLS in corso...');
31        hls.destroy();
32        break;
33      }
34    }
35   });
36  }
```

Listato 4.6: `script.js` - Codice JavaScript minimale per il corretto funzionamento di hls.js.

Adesso è possibile premere il pulsante play per riprodurre il contenuto A/V.

Un passo facoltativo nella configurazione di hls.js consiste nel dichiarare una variabile `struct`, impostare il valore dei campi relativi ai parametri di HLS che si vogliono modificare e passare `config` come parametro al costruttore dell'oggetto HLS. Nel listato 4.7 la variabile `config` viene istanziata e configurata in maniera tale per cui:

- le stampe di debug sono abilitate;
- la modalità a bassa latenza è disabilitata;
- la capienza massima dei buffer è impostata a 60 s.

```
1  var config = {
2   debug: true,
3   lowLatencyMode: false,
4   maxBufferLength: 60,
5   ...
6  }
```

```
7  ...
8  var hls = new Hls(config);
```

Listato 4.7: `script.js` - Esempio di configurazione avanzata dell'oggetto HLS.

4.1.5 Separazione delle tracce audio e video

Ora che la libreria è configurata correttamente, si procede con l'individuazione, attraverso un'analisi approfondita del codice sorgente e della documentazione di hls.js, dei moduli da modificare al fine di indirizzare le tracce audio e video del flusso verso gli elementi `<audio>` e `<video>`.

Premessa

Il codice di hls.js si appoggia ad alcune interfacce delle Web API, tipi di oggetto definiti da Mozilla Foundation, per la corretta gestione dei contenuti audio e video. In questa premessa vengono descritte le interfacce incontrate nel corso dell'analisi di fattibilità:

- `HTMLElement` rappresenta un qualsiasi elemento HTML all'interno della pagina. Alcuni elementi implementano direttamente quest'interfaccia, mentre altri implementano un'interfaccia che eredita da `HTMLElement` [25];
- `HTMLMediaElement` eredita da `HTMLElement`, aggiungendo le proprietà e i metodi necessari al supporto delle operazioni di base su contenuti audio e video [26];
- `HTMLVideoElement` e `HTMLAudioElement` ereditano da `HTMLMediaElement`, aggiungendo proprietà e metodi specifici per la manipolazione di oggetti video e audio [27, 28].

La gerarchia delle interfacce elencate finora è riportata in figura 4.3.

Inoltre, si aggiungono le seguenti interfacce:

- `MediaSource`, un'interfaccia delle MSE, che rappresenta una sorgente di dati multimediali (audio e/o video). Per riprodurre il contenuto di un oggetto di tipo `MediaSource`, si assegna quest'ultimo ad un oggetto `HTMLMediaElement` [29];

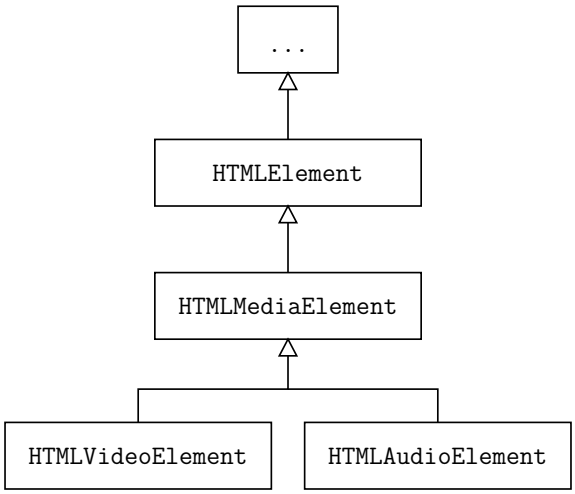

Figura 4.3: Gerarchie delle interfacce.

- `SourceBuffer`, che rappresenta un chunk di dati multimediali da assegnare ad un `HTMLMediaElement` per la riproduzione tramite un oggetto `MediaSource`. Può essere composto da uno o più buffer di byte, detti `ArrayBuffer`, di lunghezza fissa e contenenti i dati binari dei flussi multimediali. Gli `ArrayBuffer` non sono manipolabili in maniera diretta e possono contenere, ad esempio, campioni audio compressi in AAC o immagini compresse in H.264 [30, 31].

I legami tra queste interfacce sono riepilogati nello schema in figura 4.4.

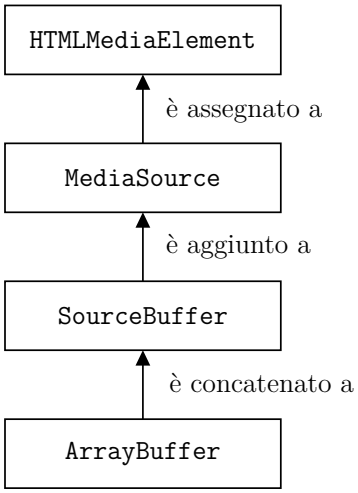

Figura 4.4: Relazioni tra le interfacce Web API per la gestione dell'A/V.

Inizializzazione delle interfacce

In questa fase si creano le interfacce per il salvataggio dei dati relativi alle tracce audio e video. I passaggi principali sono elencati di seguito e riassunti nel diagramma di flusso in figura 4.5.

1. | | |
   |---|---|
   | File | `script.js` |
   | Metodo | `attachMedia(video)` |
   | Evento | `MEDIA_ATTACHING` |

   L'`HTMLVideoElement` è legato all'istanza HLS.

2. | | |
   |---|---|
   | File | `buffer-controller.ts` |
   | Metodo | `onMediaAttaching(...)` |
   | Evento | - |

   Con riferimento al listato 4.8:

   (a) si memorizza il riferimento al `<video>` nella variabile `media` (riga 2);

   (b) si definisce il nuovo `MediaSource ms` (riga 5);

   (c) si aggancia l'evento `sourceopen` di `ms` al metodo `_onMediaSourceOpen()` (riga 7);

   (d) si assegna `ms` a `media` (riga 10).

```
1  // Salvataggio del riferimento all'HTMLVideoElement
2  let media = this.media = data.media;
3    if (media && MediaSource) {
4      // Setup del MediaSource ms
5      let ms = this.mediaSource = new MediaSource();
6      // Definizione del listener per l'evento sourceopen di ms
7      ms.addEventListener('sourceopen', this._onMediaSourceOpen);
8      ...
9      // Aggancio del video al MediaSource
10     media.src = window.URL.createObjectURL(ms);
11     ...
12   }
13 ...
```

Listato 4.8: `buffer-controller.ts` - Estratto di `onMediaAttaching()`.

3. | | |
|---|---|
| File | `buffer-controller.ts` |
| Metodo | `_onMediaSourceOpen(...)` |
| Evento | `MEDIA_ATTACHED` |

   L'evento scatenato è intercettato dal metodo `onMediaAttached()` del modulo `AudioStreamController`. Al ritorno dal suddetto metodo, viene invocato `checkPendingTracks()`, che a sua volta invoca `createSourceBuffers()`.

4. | | |
|---|---|
| File | `audio-stream-controller.js` |
| Metodo | `onMediaAttached(...)` |
| Evento | - |

   Il riferimento all'`HTMLVideoElement` viene memorizzato in una variabile dedicata.

5. | | |
|---|---|
| File | `buffer-controller.ts` |
| Metodo | `createSourceBuffers(...)` |
| Evento | `BUFFER_CREATED` |

   Ogni traccia del flusso viene assegnata ad un nuovo `SourceBuffer`. I buffer, a loro volta, sono assegnati al `MediaSource` del `<video>`.

6. | | |
|---|---|
| File | `audio-stream-controller.js` |
| Metodo | `onBufferCreated(...)` |
| Evento | - |

   I riferimenti ai nuovi `SourceBuffer` sono salvati in opportune variabili.

Elaborazione dei fragment

Una volta completata la fase iniziale, si prosegue con l'elaborazione dei segmenti della playlist, chiamati fragment nel contesto della libreria hls.js, per estrarne gli

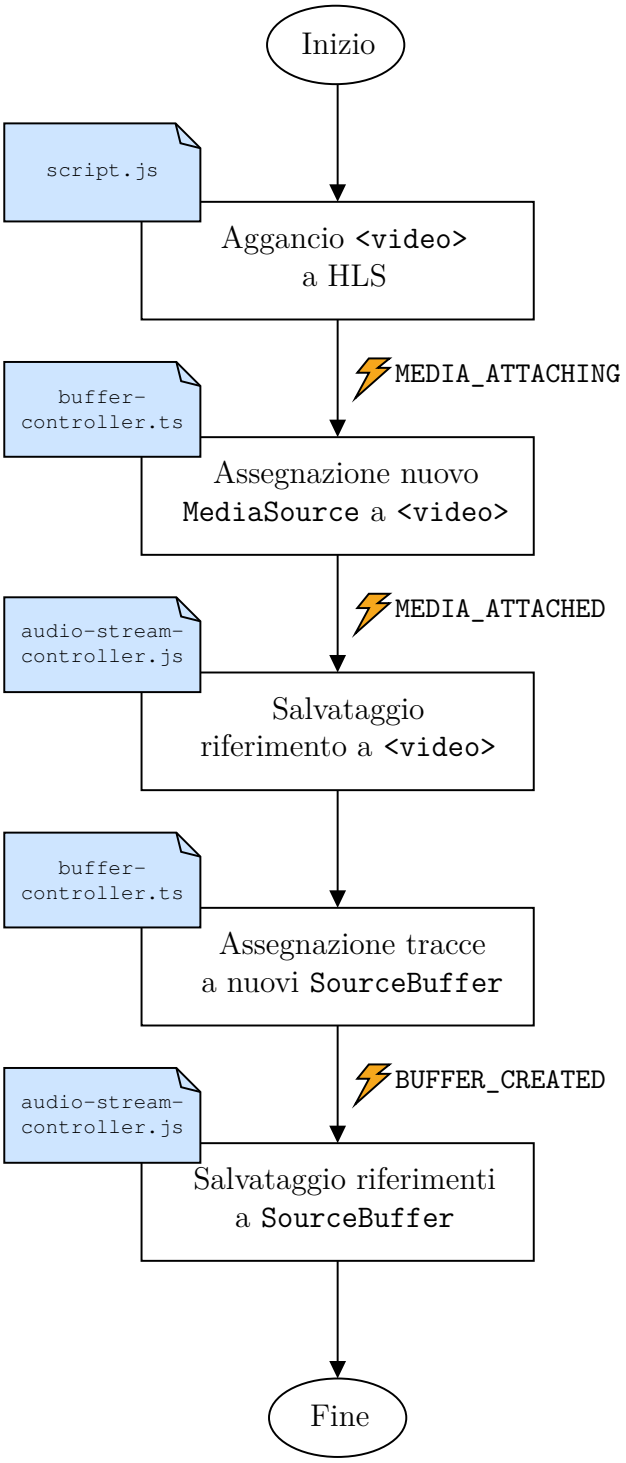

Figura 4.5: Diagramma di flusso del processo di inizializzazione delle interfacce.

`ArrayBuffer` ed inserirli nei `SourceBuffer`. I passaggi principali sono elencati di seguito e riassunti nel diagramma di flusso in figura 4.7.

1. | | |
   |---|---|
   | File | `script.js` |
   | Metodo | `loadSource(mySource)` |
   | Evento | `MANIFEST_LOADING` |

   Si indica il path o l'URL della playlist da riprodurre.

2. | | |
   |---|---|
   | File | `fragment-loader.js` |
   | Metodo | `loadsuccess(...)` |
   | Evento | `FRAG_LOADED` |

   Si effettua il download del file relativo al fragment corrente.

3. | | |
   |---|---|
   | File | `demuxer.js` |
   | Metodo | `push(...)` |
   | Evento | `FRAG_PARSING_INIT_SEGMENT` |

   La prima volta che questo metodo è invocato, si istanzia il demuxer, un component che effettua la separazione delle tracce all'interno del fragment (come mostrato in figura 4.6).

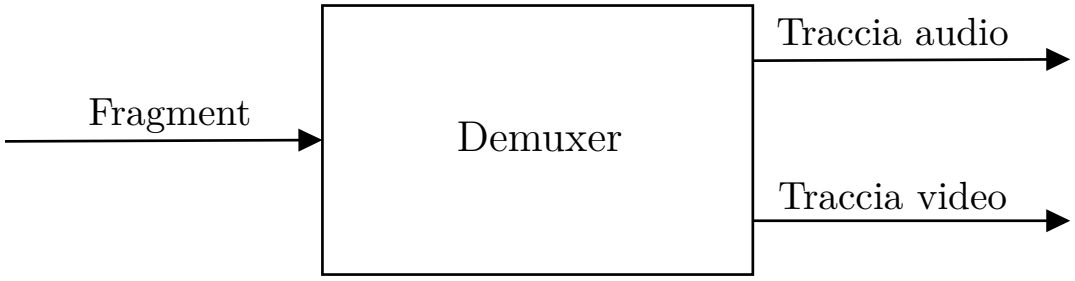

Figura 4.6: Separazione delle tracce di un fragment tramite il demuxer.

4. | | |
   |---|---|
   | File | `audio-stream-controller.js` |
   | Metodo | `onFragParsingInitSegment(...)` |
   | Evento | `BUFFER_CODECS` |

I codec utilizzati per le tracce vengono estratti dal fragment e utilizzati per l'inizializzazione dei `SourceBuffer`.

5. | | |
   |---|---|
   | File | `buffer-controller.ts` |
   | Metodo | `doAppending()` |
   | Evento | `BUFFER_APPENDED` |

   Utilizzando le informazioni contenute nei codec si inseriscono in append i dati relativi alle tracce nei `SourceBuffer` dedicati.

6. Se il fragment corrente non è l'ultimo della playlist, si ripetono gli step 2÷5 per il fragment successivo, altrimenti l'operazione è conclusa.

Aggiunte al codice

L'unica aggiunta ad alto livello, ossia nel contesto del player, consiste nel creare un `HTMLAudioElement` aggiungendo un elemento `<audio>` nella pagina HTML e assegnandogli un identificativo (listato 4.9, riga 7).

```
1  <!DOCTYPE html>
2  <html>
3    <head>...</head>
4    <body>
5      ...
6      <!-- Elemento audio -->
7      <audio id="audio"></audio>
8      ...
9    </body>
10 </html>
```

Listato 4.9: `index.html` - Introduzione dell'`HTMLAudioElement`.

Si elencano di seguito le aggiunte effettuate al modulo `BufferController` di hls.js:

1. nel costruttore vengono dichiarati un `SourceBuffer` e un `MediaSource` per i dati relativi all'audio (listato 4.10, righe 4 e 6);

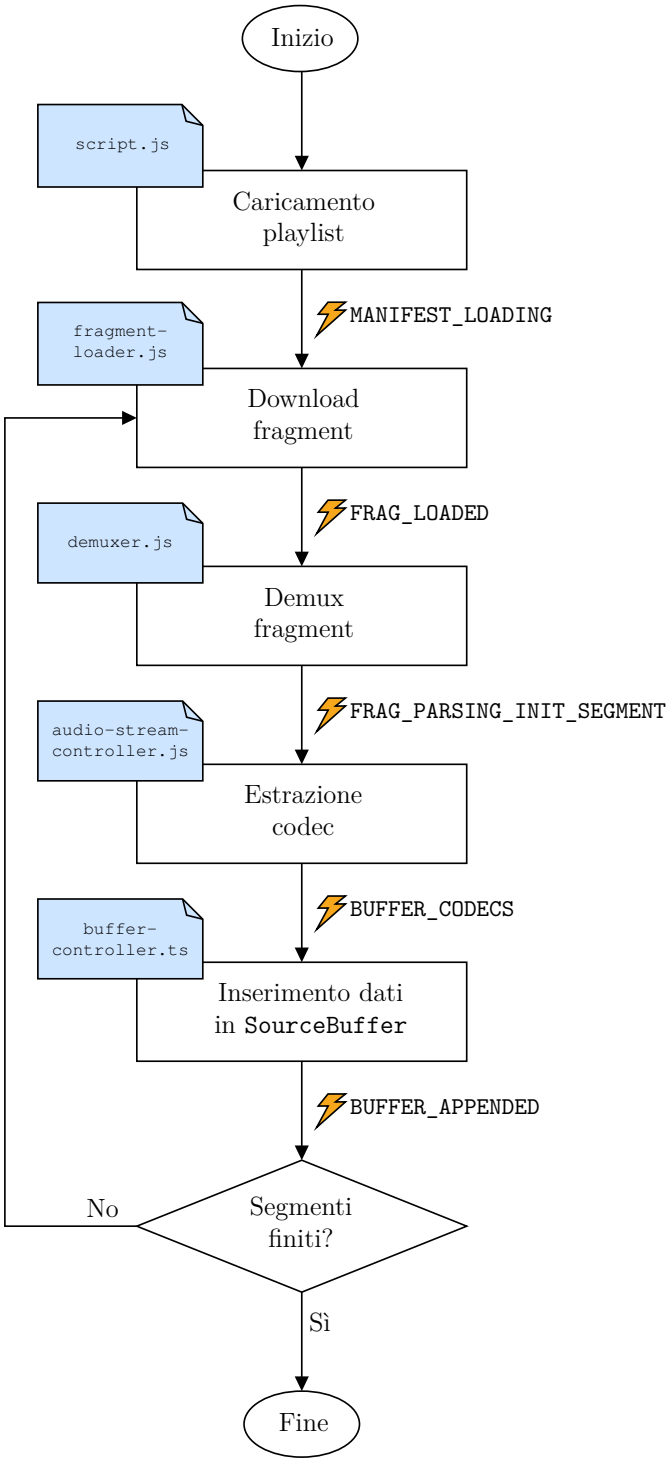

Figura 4.7: Diagramma di flusso del processo di elaborazione dei fragment.

```
1  class BufferController extends EventHandler {
2    ...
3    // Nuovo SourceBuffer per <audio>
4    public sb_audio: SourceBuffer = {};
5    // Nuovo MediaSource per <audio>
6    public ms_audio: MediaSource | null = null;
7    ...
8  }
```

Listato 4.10: `buffer-controller.ts` - Aggiunte nel costruttore.

2. nel metodo `onMediaAttaching()`, in maniera del tutto analoga a quanto descritto per il `MediaSource ms` nel listato 4.8, vengono eseguite le seguenti operazioni:

   (a) si memorizza il riferimento all'`<audio>` nella variabile `audioElem` (riga 7);

   (b) si inizializza il `MediaSource ms_audio` (riga 9);

   (c) si aggancia l'evento `sourceopen` di `ms_audio` al metodo introdotto nello step 3, `_onMediaSourceOpen_audio()` (riga 14);

   (d) si assegna `ms_audio` a `audioElem` (riga 16);

```
1  onMediaAttaching (data: { media: HTMLMediaElement }) {
2    // Salvataggio del riferimento all'HTMLVideoElement
3    let media = this.media = data.media;
4    if (media && MediaSource) {
5      ...
6      // Salvataggio del riferimento all'HTMLAudioElement
7      let audioElem = document.getElementById("audio");
8      // Setup di ms_audio
9      this.ms_audio = new MediaSource();
10     this.ms_audio._object = this;
11     // Setup di sb_audio
12     this.sb_audio = undefined;
13     // Definizione del listener per l'evento sourceopen di ms_audio
```

```
14      this.ms_audio.addEventListener('sourceopen',
          this.onMediaSourceOpen_audio);
15      // Aggancio dell'audio al MediaSource
16      audioElem.src = URL.createObjectURL(this.ms_audio);
17    }
18  }
```

Listato 4.11: `buffer-controller.ts` - Aggiunte in `onMediaAttaching()`.

3. si definisce `_onMediaSourceOpen_audio()`. Con riferimento al listato 4.12, questo nuovo metodo:

   (a) inizializza `sb_audio` con le informazioni del codec precedentemente estratto (riga 5);

   (b) rimuove il listener per l'evento `sourceopen` di `ms_audio` al termine dell'operazione (riga 9);

```
1   this._object = BufferController;
2   private onMediaSourceOpen_audio(){
3     if(!this._object.sb_audio){
4       // Setup di sb_audio con le info del codec
5       this._object.sb_audio =
          this._object.ms_audio.addSourceBuffer(/*Info codec*/);
6     }
7     if (this._object.ms_audio) {
8       // Rimozione del listener per l'evento sourceopen di ms_audio
9       this._object.ms_audio.removeEventListener('sourceopen',
          this._object.onMediaSourceOpen_audio);
10    }
11  }
```

Listato 4.12: `buffer-controller.ts` - `onMediaSourceOpen_audio()`.

4. nel metodo `doAppending()` i dati del segmento audio corrente vengono inseriti in append nel `SourceBuffer sb_audio`.

```
1   doAppending () {
2     // Recupero del segmento
3     const segment = ...;
```

```
4     ...
5     try {
6       ...
7       // Se il segmento contiene audio...
8       if(segment.type === 'audio'){
9         ...
10        // ...concatena i suoi dati in sb_audio
11        this.sb_audio.appendBuffer(segment.data);
12      }
13    }
14    ...
15  }
```

Listato 4.13: `buffer-controller.ts` - Aggiunte in `doAppending()`.

A questo punto la traccia audio del flusso è correttamente indirizzata verso l'elemento `<audio>` della pagina HTML e l'elemento `<video>` riceve esclusivamente la traccia video.

4.1.6 Definizione di metatag custom

Classificazione dei metatag

Nel sottoparagrafo 3.3.3 si è posto l'accento sulla robustezza di HLS a fronte di metatag aggiunti dall'utente e non previsti dal protocollo. A tal proposito, è possibile classificare i metatag di HLS in:

- globali, che contengono informazioni generali sulla playlist. Alcuni esempi di metatag globali sono:
  - `#EXT-X-TARGETDURATION`;
  - `#EXT-X-MEDIA-SEQUENCE`;
  - `#EXT-X-ENDLIST`;
- locali, relativi al singolo segmento della playlist. Alcuni esempi di metatag locali sono:
  - `#EXTINF`;

- `#EXT-X-BYTERANGE`;
- `#EXT-X-DISCONTINUITY`.

Il listato 4.14 mostra un esempio di media playlist con l'aggiunta di alcuni metatag globali e locali di prova. Il codice è stato indentato per meglio distinguere gli scope locali dei segmenti (tra cui lo scope del segmento `segment001.ts`, evidenziato in grassetto) dallo scope globale.

```
#EXTM3U
...
#EXT-GLOBAL-METATAG-1:value
 #EXTINF:10.000,
 #EXT-LOCAL-METATAG-1:value
 segment001.ts
 #EXTINF:10.000,
 #EXT-LOCAL-METATAG-1:value
 #EXT-LOCAL-METATAG-2:value
 segment002.ts
...
#EXT-GLOBAL-METATAG-2:value
#EXT-GLOBAL-METATAG-3:value
 #EXTINF:10.000,
 segment099.ts
...
#EXT-GLOBAL-METATAG-4:value
#EXT-X-ENDLIST
```

Listato 4.14: Media playlist con metatag globali e locali di prova.

Aggiunte al codice

Sono elencate di seguito le aggiunte apportate al codice di hls.js:

1. si modifica il modulo `M3U8Parser` in modo tale che il parser della libreria riconosca i nuovi metatag (listato 4.15);

```
/* Nuovi metatag da riconoscere:
    - EXT-GLOBAL-METATAG-1
    - EXT-GLOBAL-METATAG-2
```

```
4       - EXT-LOCAL-METATAG-1
5       - EXT-LOCAL-METATAG-2
6   */
7   const LEVEL_PLAYLIST_REGEX_SLOW = new RegExp(...);
```

Listato 4.15: `m3u8-parser.ts` - Aggiunte al parser.

2. si dichiarano i nuovi eventi, elencati nel modulo `Events`, da scatenare quando il parser riconosce i nuovi metatag (listato 4.16);

```
1   const HlsEvents = {
2     ...
3     GLOBAL_METATAG_1_LOADED: 'hlsGlobalMetatag1Loaded',
4     GLOBAL_METATAG_2_LOADED: 'hlsGlobalMetatag2Loaded',
5     LOCAL_METATAG_1_LOADED: 'hlsLocalMetatag1Loaded',
6     LOCAL_METATAG_2_LOADED: 'hlsLocalMetatag2Loaded'
7   };
```

Listato 4.16: `events.js` - Dichiarazione dei nuovi eventi da intercettare.

A questo punto si ritorna al file di alto livello `myScript.js` per introdurre i listener degli eventi relativi ai nuovi metatag (listato 4.17). I dati, assegnati alla variabile `data`, possono essere stampati sulla console del browser (riga 3) o mostrati in un elemento HTML dedicato (riga 8).

```
1   hls.on(Hls.Events.GLOBAL_METATAG_1_LOADED, function(event, data) {
2    /* EXT-GLOBAL-METATAG-1 intercettato */
3    console.log('GLOBAL_METATAG_1: ' + data);
4   });
5   ...
6   hls.on(Hls.Events.LOCAL_METATAG_1_LOADED, function(event, data) {
7    /* EXT-LOCAL-METATAG-1 intercettato */
8    document.getElementById('localMetatag1_textArea').innerHTML += data;
9   });
10  ...
```

Listato 4.17: `myScript.js` - Introduzione dei listener dei nuovi eventi.

4.2 Integrazione

Una volta terminata la fase di analisi di fattibilità, si prosegue con la fase di integrazione. I nuovi obiettivi prefissati sono i seguenti:

- integrare i canali video e dati nel player aziendale;
- gestire i comandi per il video;
- implementare alcune funzionalità aggiuntive, tra cui l'applicazione di filtri visivi e la cattura dei fotogrammi del video, per rendere più agevole l'uso del player.

4.2.1 Il player allo stato attuale

Architettura

Il player è un'applicazione web che riproduce solo flussi audio ed è creata con React [32], un framework open-source sviluppato da Facebook e scritto in JavaScript. Il codice di React è regolato da entità simili a funzioni JavaScript, denominate component, che accettano dei dati detti props (abbreviazione dell'inglese properties) e restituiscono la descrizione di ciò che deve apparire su schermo sotto forma di elementi React. Inoltre, il framework mette a disposizione alcune funzioni speciali chiamate hooks: ad esempio, l'hook `useState` permette di aggiungere uno stato al component, mentre l'hook `useEffect` è usato per indicare l'azione (ossia l'effect) che un component deve compiere dopo essere stato renderizzato.

Il player è integrato in un ambiente basato sul framework Angular [33] di Google. Poiché ogni framework segue il proprio standard, l'utilizzo combinato di React e Angular impedisce il riuso di frammenti di codice. La soluzione a questo problema è rappresentata dai Web Components [34], una serie di API standardizzate nel 2012. I Web Components fungono da "collante" tra React e Angular, poiché consentono di creare component HTML riutilizzabili dai diversi framework. Inoltre, è possibile isolare i component tramite lo Shadow DOM [35], un albero DOM nascosto i cui elementi vengono renderizzati indipendentemente dal DOM principale.

La libreria wavesurfer.js

Una delle caratteristiche più interessanti del player audio è l'integrazione di wavesurfer.js [36], una libreria JavaScript basata sulle WebAudioAPI e CanvasAPI. Dati una sorgente audio ed un elemento `<div>`, wavesurfer.js produce la forma d'onda dell'audio e la disegna nel `<div>`. Vengono messi a disposizione una serie di metodi, opzioni ed eventi per la personalizzazione delle funzionalità e dell'aspetto della forma d'onda visualizzata. Di base wavesurfer.js disegna l'intera forma d'onda del file audio, ma gli sviluppatori dell'azienda, in base alle proprie esigenze, hanno modificato il codice sorgente della libreria per dividere la forma d'onda in frammenti di ugual durata (pari a quella dei segmenti audio in arrivo al player) e disegnarli sequenzialmente.

La figura 4.8 mostra il player allo stato attuale.

Figura 4.8: Il player audio allo stato attuale.

4.2.2 Integrazione del canale video

La parte di codice in cui bisogna effettuare l'integrazione è quella scritta in React. Dopo un'analisi preliminare del codice esistente si può delineare una gerarchia dei component su cui è costruito il player attuale (figura 4.9).

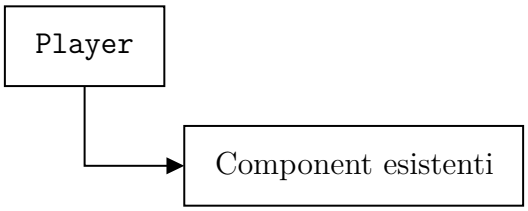

Figura 4.9: Gerarchia dei component del player audio allo stato dell'arte.

Si comincia con la definizione del component `VideoZone`, che estende `Player` (figura 4.10) e rappresenta la radice di tutte le modifiche da effettuare. Strutturalmente

corrisponde ad un `<div>`, posizionato al di sopra del player attuale, in cui bisogna inserire tutti gli elementi.

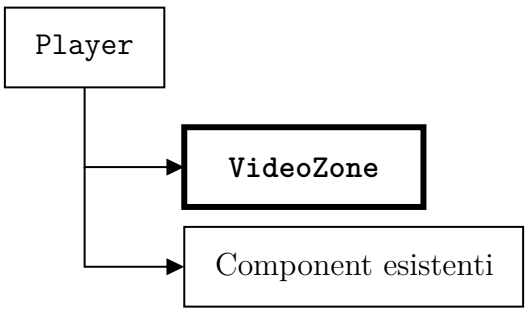

Figura 4.10: Gerarchia del player con l'aggiunta del component `VideoZone`.

Si riportano i passaggi principali per l'integrazione delle funzionalità di base di hls.js:

1. si definiscono gli elementi HTML `<video>` e `<audio>`;

2. si installa hls.js (tramite il gestore di pacchetti `npm` o `yarn`) e si modifica il codice sorgente come mostrato nella fase di analisi di fattibilità, facendo attenzione al fatto che gli elementi sopra definiti non compaiono nel DOM principale bensì nello Shadow DOM. Nel listato 4.18 è mostrato il recupero dell'elemento `<audio>` dallo Shadow DOM;

```
1 let audioElem;
2 audioElem = document.querySelector("#audio");
3 if(audioElem == null){
4   // L'elemento audio è nello shadow DOM
5   audioElem = document.shadowRoot.querySelector("#audio");
6 }
```

Listato 4.18: `buffer-controller.ts` - Controllo aggiuntivo per elemento `<audio>` nello Shadow DOM.

3. si introduce un hook `useEffect` in cui si riporta il codice del listato 4.6.

Prende così forma la gerarchia mostrata in figura 4.11.

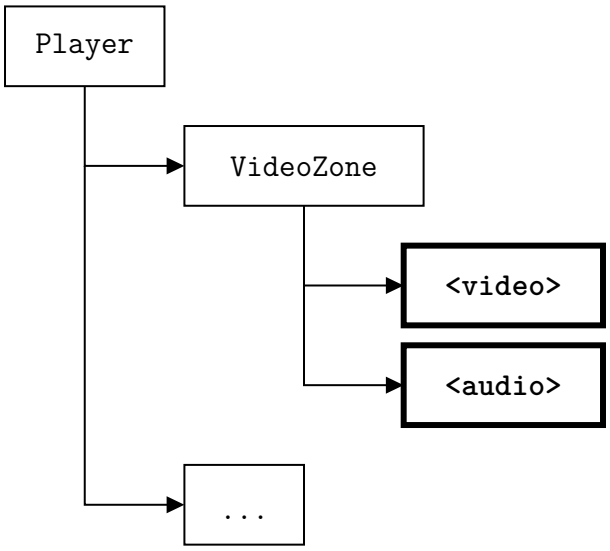

Figura 4.11: Gerarchia del player con l'aggiunta degli elementi `<video>` e `<audio>`.

4.2.3 Integrazione del canale dati

Si definisce il primo component che estende `VideoZone`: `PlaylistInfo`. Esso contiene le informazioni relative ai metatag di HLS definiti dall'utente. Si definisce anche il component `CustomDataInfo`, che offre varie possibilità per una visualizzazione efficace delle informazioni aggiuntive.

Coordinate GPS

La latitudine e la longitudine inviate dal trasmettitore sono date in input ad una mappa che mostri la posizione e gli spostamenti del dispositivo. Per questo lavoro si sceglie di utilizzare Leaflet [37], una libreria open-source scritta in JavaScript altamente personalizzabile. In maniera molto simile a Google Maps, Leaflet consente di spostarsi sulla mappa, posizionare segnaposti e regolare il livello di zoom (figura 4.12).

Stato della rete

La qualità del segnale di rete e la capacità del canale possono essere visualizzate tramite un VU meter digitale (figura 4.13).

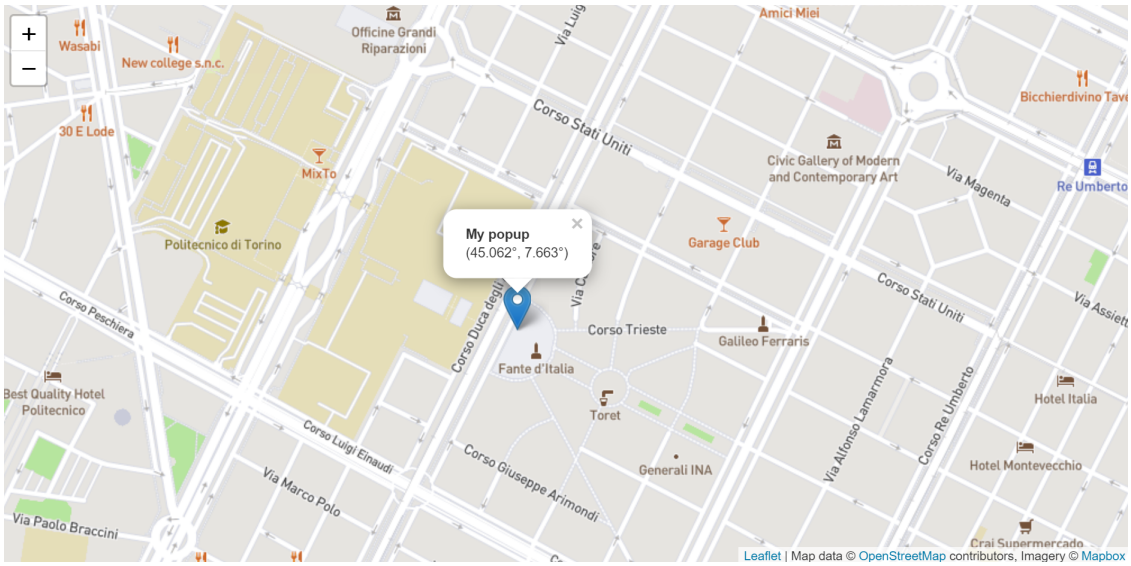

Figura 4.12: Esempio di mappa sviluppata con Leaflet.

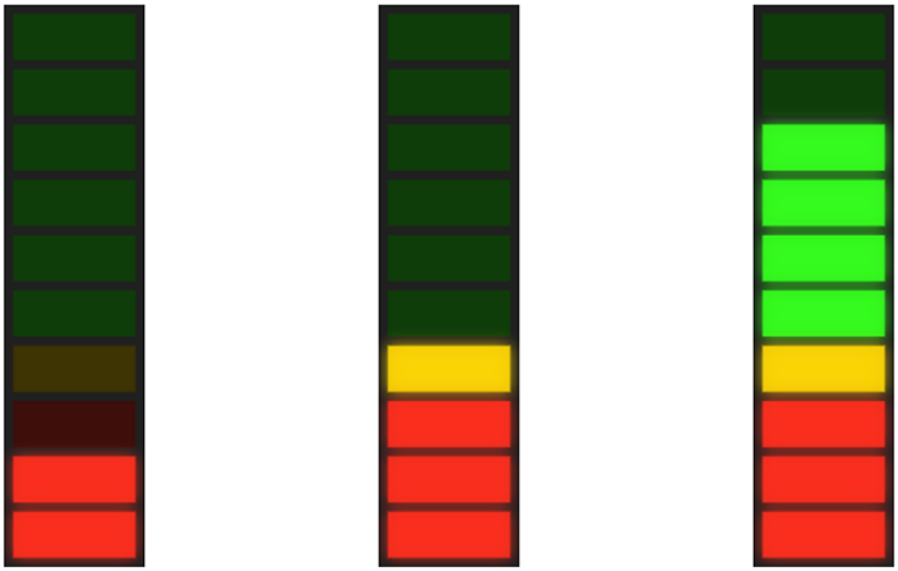

Figura 4.13: Visualizzazione dello stato della rete mediante VU meter.

Livello di batteria

Il livello della batteria del trasmettitore pilota un costrutto `switch` (listato 4.19) per mostrare le diverse varianti dell'icona di una batteria (figura 4.14).

```
switch(true){
 case batteryLevel <= 10:
  // Intervallo 0%-10%: mostra icona BatteryAlert
  break;
 case batteryLevel <= 20:
  // Intervallo 11%-20%: mostra icona Battery20
  break;
 case batteryLevel <= 50:
  // Intervallo 21%-50%: mostra icona Battery50
  break;
 case batteryLevel <= 80:
  // Intervallo 51%-80%: mostra icona Battery80
  break;
 case batteryLevel <= 100:
  // Intervallo 81%-100%: mostra icona BatteryFull
  break;
}
```

Listato 4.19: Costrutto `switch` per la visualizzazione dell'icona più adatta al livello di batteria corrente.

Figura 4.14: Visualizzazione del livello di batteria mediante icone opportune.

La figura 4.15 mostra la gerarchia aggiornata dei component.

4.2.4 Gestione dei comandi per il video

Quest'obiettivo consiste nell'aggiungere una barra dei controlli nella parte inferiore del video in riproduzione per ottenere così il layout tipico di un player multimediale. Esistono numerose librerie open-source disponibili in rete che consentono di raggiungere quest'obiettivo e la scelta per questo lavoro è ricaduta su video.js [38].

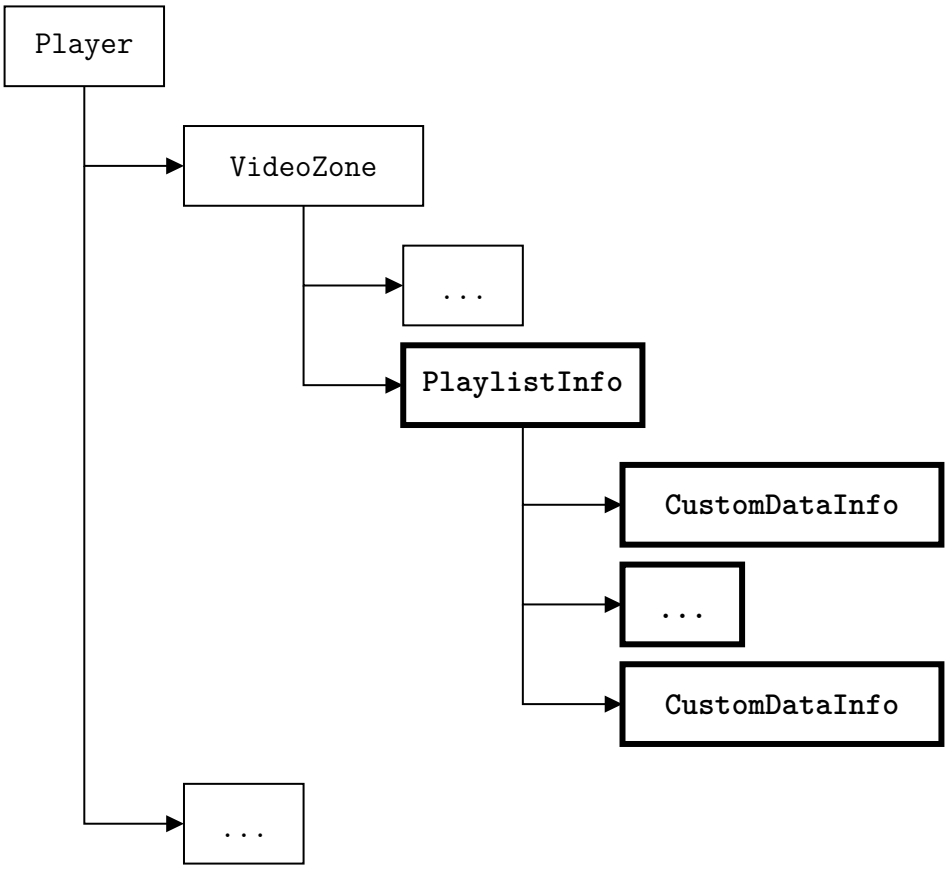

Figura 4.15: Gerarchia del player con l'aggiunta dei component `PlaylistInfo` e `CustomDataInfo`.

Questa libreria JavaScript consente di costruire un player attorno a un elemento video HTML5 (figura 4.16) e mette a disposizione vari moduli per la personalizzazione sia dello stile (tramite skins) che delle funzionalità (tramite components). Il video da riprodurre può essere sia un flusso in formato "tradizionale" (come MP4 o WebM) che un flusso per lo streaming adattivo (gestito dai protocolli HLS o MPEG-DASH). Inoltre, video.js è supportata dalla maggior parte dei browser attuali, sia desktop che mobile.

Si definisce il component `VideojsManagement`, che estende `VideoZone`, per la gestione del player di video.js. Dal momento che nel player attuale sono già presenti i pulsanti per la riproduzione dell'audio, si modificano i rispettivi handler affinché essi gestiscano anche la riproduzione del video: in questo modo, la control bar non necessita né di questi pulsanti né di una barra di avanzamento. Lo stesso vale anche

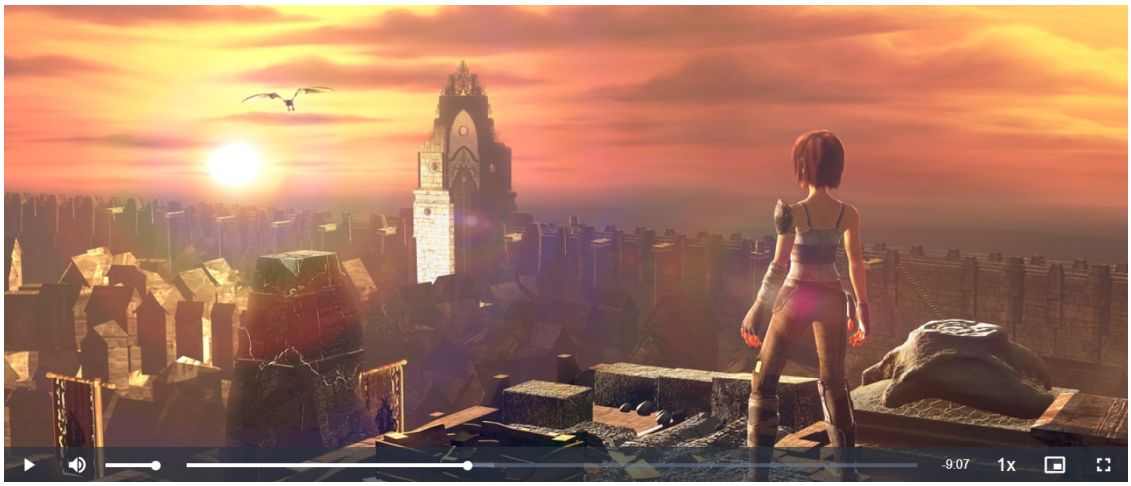

Figura 4.16: Layout standard della control bar del player di video.js.

per lo slider per la regolazione del volume (figura 4.17).

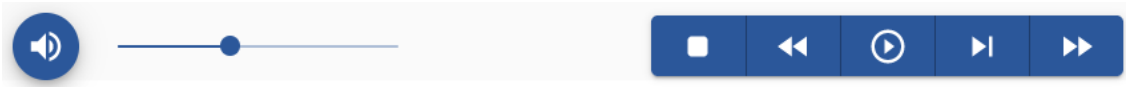

Figura 4.17: Dettaglio dei controlli per il volume e la riproduzione dell'audio del player.

Facendo riferimento alla pagina HTML d'esempio in figura 4.18, in cui sono stati inseriti alcuni elementi `<div>` e il player di video.js, si creano due component, che estendono `VideojsManagement`, relativi ai seguenti pulsanti da mostrare nella control bar:

- il `FullscreenButton` per attivare la visualizzazione del video a schermo intero (figura 4.19);
- il `PictureInPictureButton` per attivare una modalità Picture-in-Picture personalizzata in cui il riquadro del video è sganciato dalla pagina e può essere trascinato e ridimensionato (figura 4.20). Una novità rispetto alla PiP nativa è la possibilità di fissare una dimensione di soglia per la larghezza della finestra del browser: così, quando l'utente restringe la finestra fino a raggiungere la soglia la modalità PiP viene attivata automaticamente, mentre essa viene disattivata quando l'utente riporta la finestra ad una larghezza superiore alla soglia.

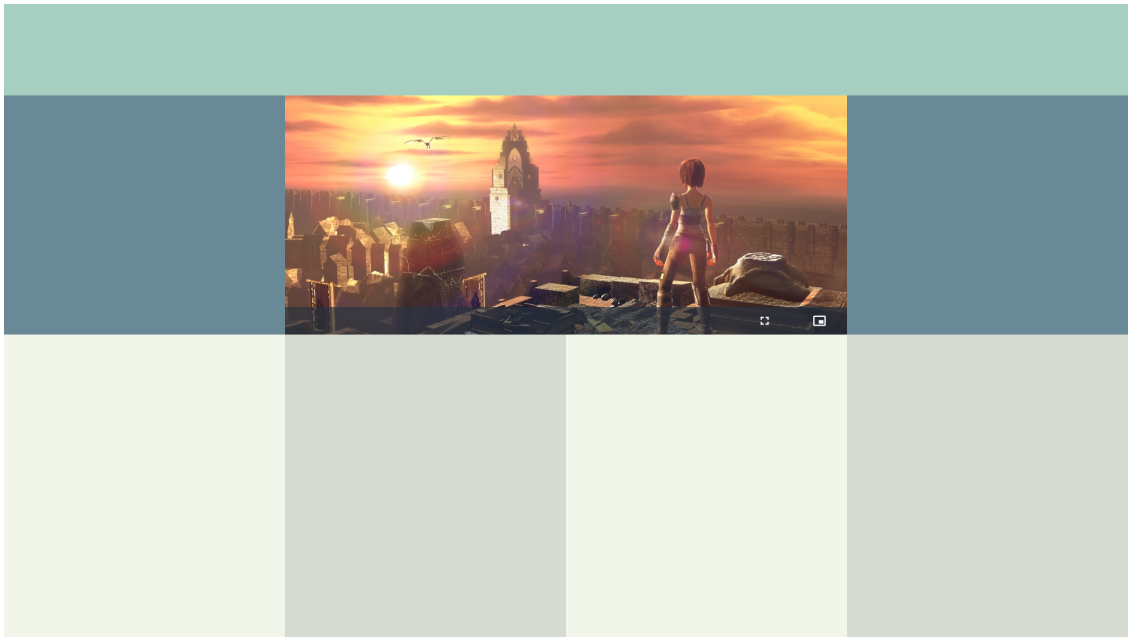

Figura 4.18: Il player di video.js all'interno di una pagina d'esempio.

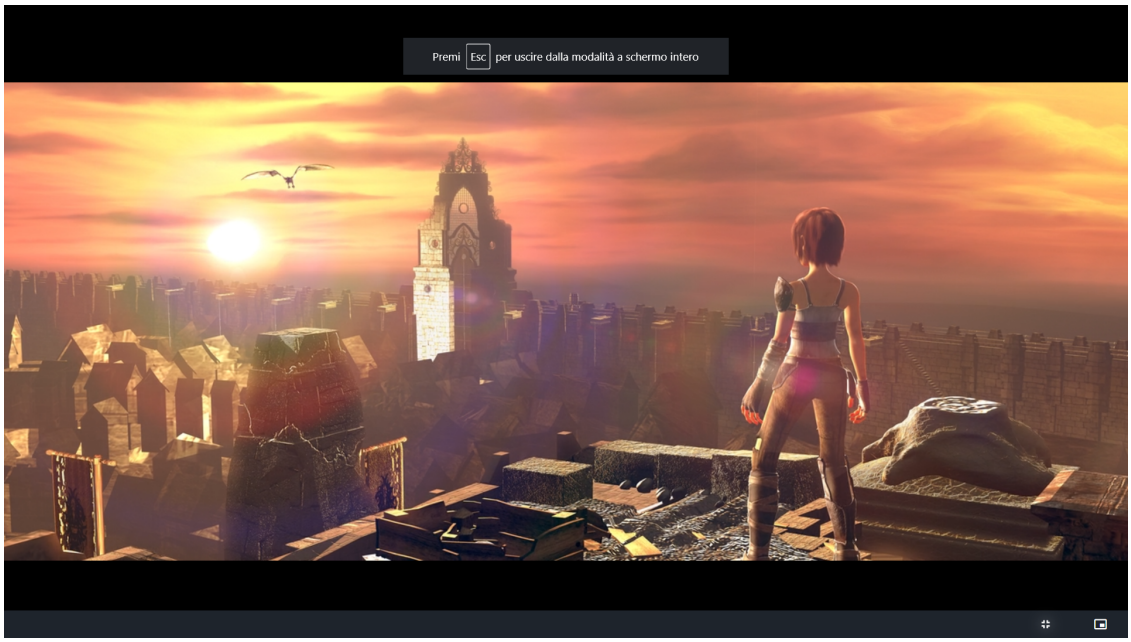

Figura 4.19: Il player di video.js in modalità fullscreen.

La nuova control bar appare come in figura 4.21 e la gerarchia aggiornata dei component del player è mostrata in figura 4.22.

Figura 4.20: Il player di video.js in modalità PiP.

Figura 4.21: Il nuovo layout della control bar del player di video.js.

4.2.5 Filtri video

Si vuole dare la possibilità all'utente di applicare alcuni effetti al video tramite la regolazione della luminosità, del contrasto, della tinta etc. Per raggiungere questo obiettivo si ricorre a glfx.js [39], una libreria open-source JavaScript che mette a disposizione diversi filtri da applicare alle immagini caricate dall'utente (come

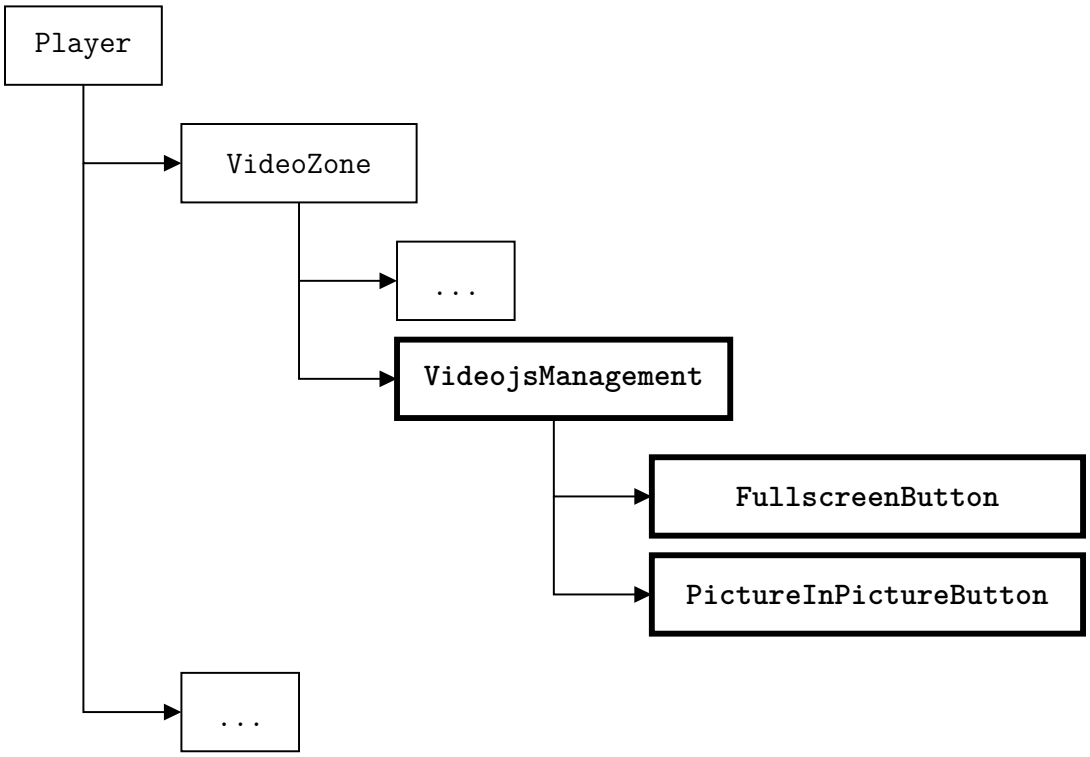

Figura 4.22: Gerarchia del player con l'aggiunta dei component per la gestione di video.js.

mostrato in figura 4.23).

Poiché il codice nativo di glfx.js non consente di modificare direttamente elementi `<video>`, sono necessari i seguenti passaggi (riassunti nello schema in figura 4.24):

- i fotogrammi del video vengono disegnati in un canvas nascosto tramite il metodo `drawImage()`. Per disegnare ciascun fotogramma del flusso si utilizza `requestAnimationFrame()`, un metodo per la gestione delle animazioni all'interno del browser che, rispetto ad altre soluzioni, consente di migliorare le perfomance e contenere l'uso delle risorse del sistema;
- glfx.js converte il canvas in texture, applica i filtri indicati dall'utente e disegna un secondo canvas.

A questo punto il secondo canvas generato viene sovrapposto al video originale del player di video.js. Per quel che concerne gli slider relativi ai filtri, si aggiunge il pulsante `VideoSettingsButton` alla control bar del video: al suo click viene

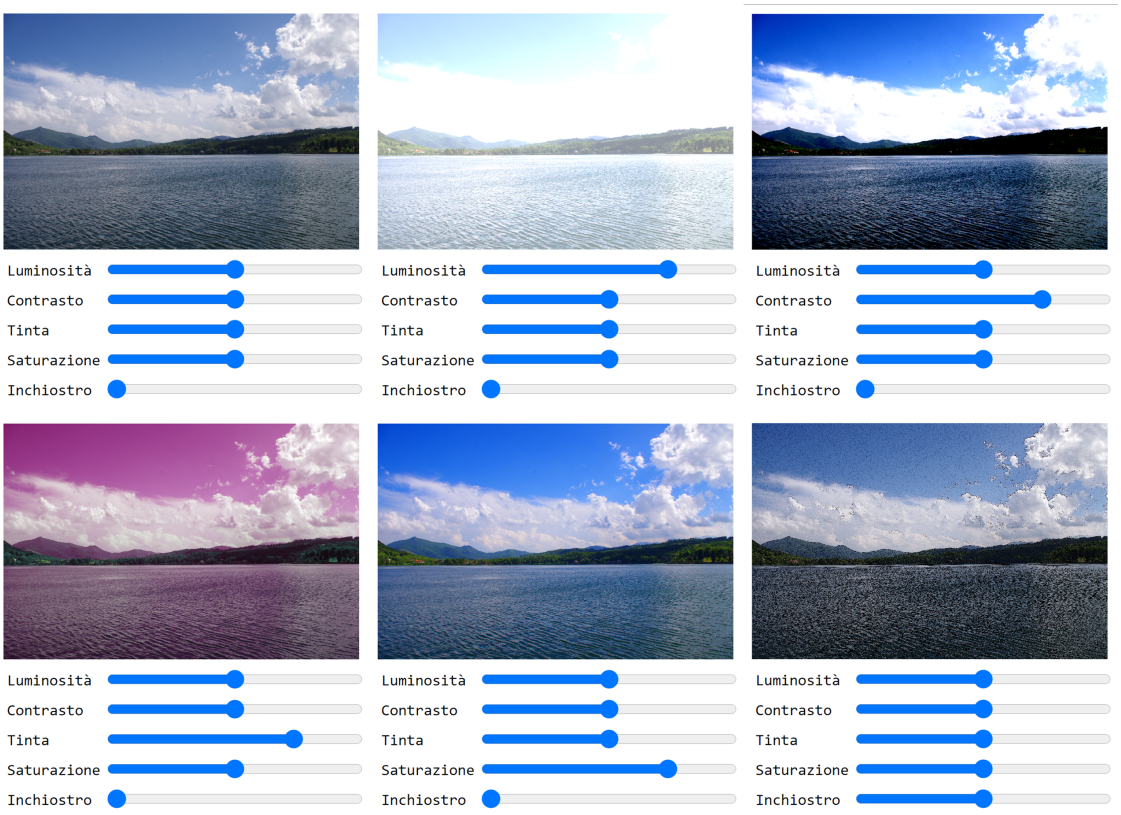

Figura 4.23: Applicazione di alcuni effetti di glfx.js su un'immagine.

visualizzato `VideoSettingsModalDialog`, un modulo di tipo modal dialog messo a disposizione da video.js, contenente gli slider dei filtri. La logica dei filtri viene gestita dal component `GlfxManagement`, che estende `VideoSettingsModalDialog` (figura 4.25).

La figura 4.26 riporta la gerarchia dei component con le ultime aggiunte.

4.2.6 Cattura dei fotogrammi

Una funzionalità utile nel contesto delle LI ambientali è rappresentata dalla cattura dei fotogrammi del video riprodotto dal player. Facendo riferimento allo schema in figura 4.27, si aggiunge un'opzione in `VideoSettingsModalDialog` (ad esempio una combobox) per scegliere il `<canvas>` sorgente da cui partire per la creazione del `<canvas>` relativo al fotogramma corrente (ossia `<canvas #canvas_snapshot>`).

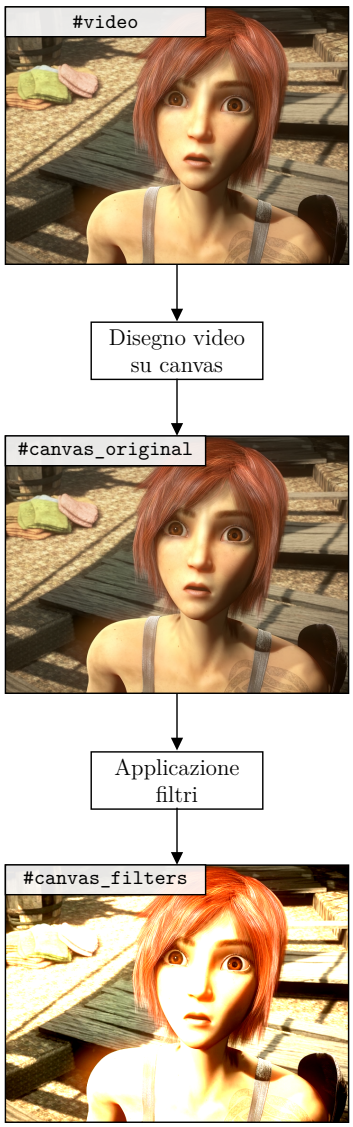

Figura 4.24: Schema del funzionamento di glfx.js.

Si introduce il component `SnapshotButton`, che estende `VideojsManagement`. Esso è un pulsante posizionato nella control bar del player (come mostrato in figura 4.28) e, quando viene cliccato, genera un nuovo `<canvas>` per il fotogramma del video all'istante corrente. A questo punto l'utente può scegliere di scaricare il fotogramma, eliminarlo o consultare le informazioni ad esso associate, come ad esempio il timestamp.

La gerarchia dei component del player, una volta implementata la cattura dei

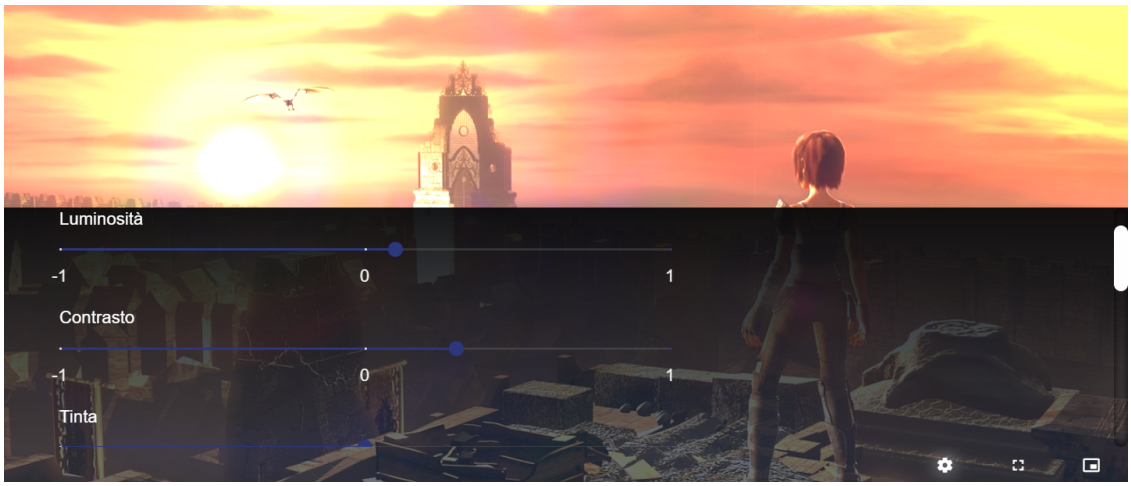

Figura 4.25: La nuova control bar del player di video.js con il modal dialog contenente gli slider per i filtri di glfx.js.

fotogrammi, appare come in figura 4.29.

4.2.7 Parametri di alto livello

Nel codice sorgente del player è presente un file di alto livello in cui sono definiti alcuni parametri da passare ai component preesistenti dell'applicazione. La loro modifica mentre il player è in esecuzione comporta il ricaricamento automatico della pagina. Si possono dunque dichiarare parametri simili anche per il component `VideoZone` e i suoi discendenti.

`customHlsTags` (Object)

Indica quali informazioni relative alle estensioni di HLS devono essere visualizzate nel component `PlaylistInfo`. Inoltre, tramite il valore del campo `showHlsTags` è possibile dire se il component `PlaylistInfo` deve essere mostrato oppure nascosto. Nell'esempio del listato 4.20 il parametro è impostato in maniera tale che `PlaylistInfo` mostri le informazioni relative al GPS e alla batteria (campi `gps` e `battery`) e nasconda quelle relative alla rete (campo `network`).

```
1  let customHlsTags = {
2   showHlsTags: true,
3   gps: true,
```

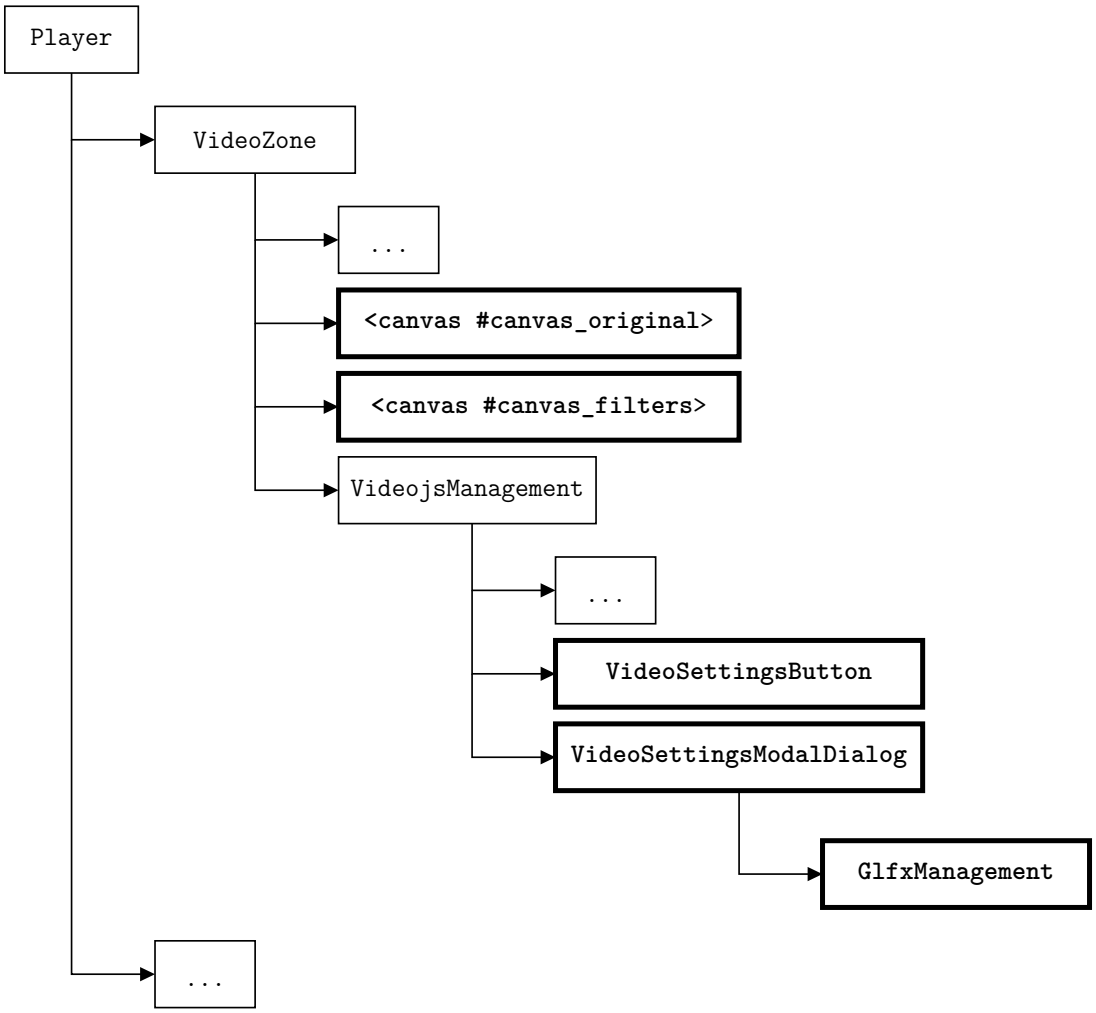

Figura 4.26: Gerarchia del player con l'aggiunta degli elementi e dei component per la gestione dei filtri di glfx.js.

```
4   network: false,
5   battery: true
6 }
```

Listato 4.20: Esempio di dichiarazione del parametro `customTags`.

`playlist` (String)

Indica la playlist M3U8 sorgente del player. Sono accettate sia playlist fornite da un server remoto che playlist locali.

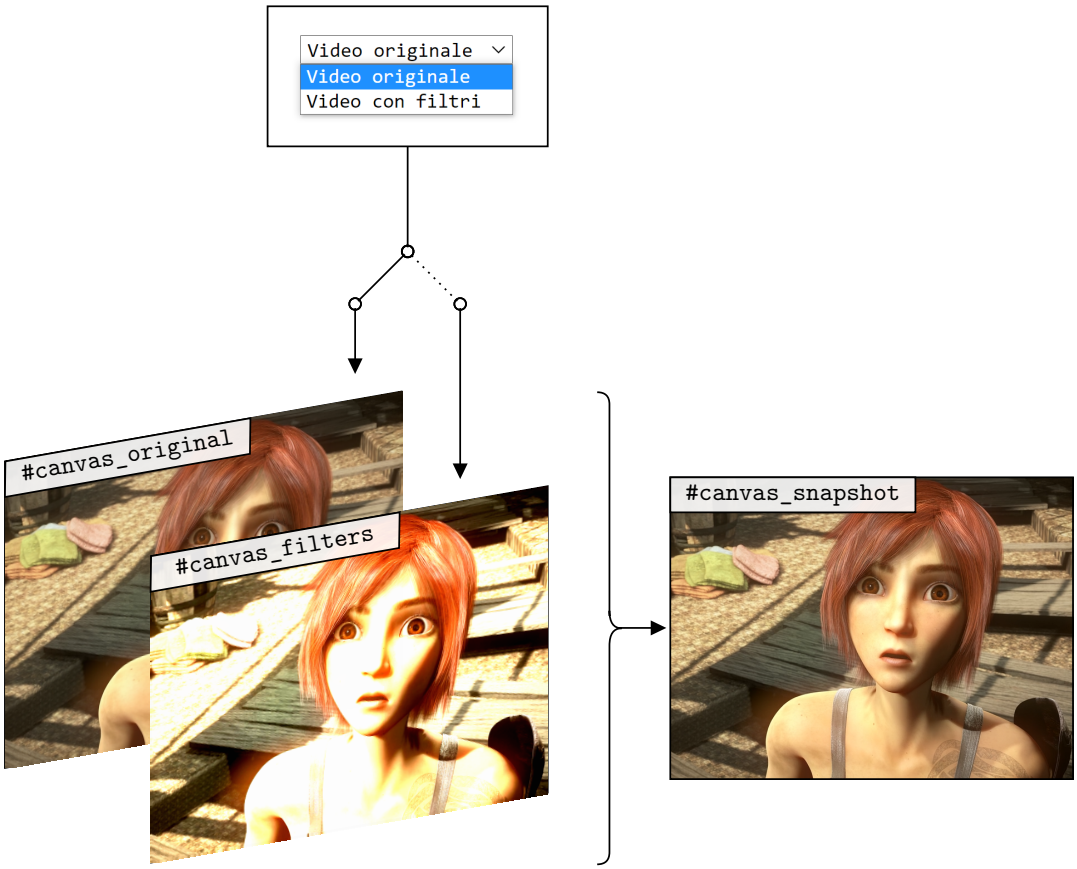

Figura 4.27: Schema del processo di scelta del `<canvas>` sorgente per l'estrazione del fotogramma.

`videoBackgroundColor` (String)

Indica il colore da applicare al `<div>` che contiene il player di video.js.

`videoWidth` (String)

Indica la larghezza (in percentuale) da applicare al player di video.js.

`showFullscreenButton` (boolean)

Indica se mostrare o nascondere il pulsante della modalità a schermo intero nella control bar del player di video.js.

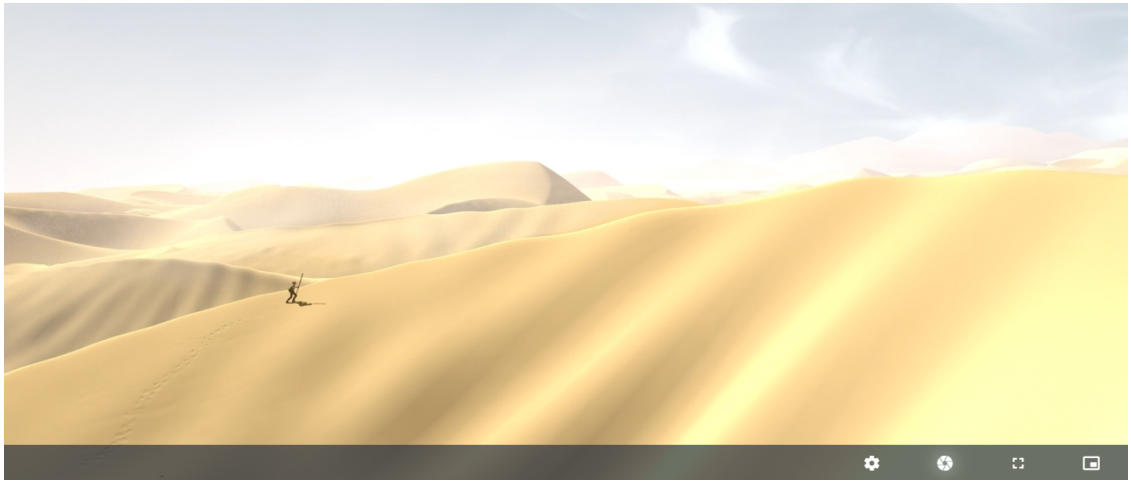

Figura 4.28: Layout della control bar del player di video.js con l'aggiunta di `SnapshotButton`.

`pipMode` (String)

Indica come viene abilitata e disabilitata la modalità PiP del video. Si faccia riferimento alla tabella 4.1 per i valori che il parametro può assumere. Quando `pipMode = noPip`, il `PictureInPictureButton` è assente e non è possibile abilitare la PiP.

Valore di `pipMode`	PiP abilitata con...	PiP disabilitata con...
`noPip`	-	-
`enterC_exitC`	C	C
`enterR_exitR`	R	R
`enterR_exitC`	R	C
`enterR_exitCR`	R	C, R
`enterCR_exitC`	C, R	C
`enterCR_exitR`	C, R	R
`enterCR_exitCR`	C, R	C, R

Tabella 4.1: Elenco dei valori ammessi per il parametro `pipMode` e delle operazioni che abilitano/disabilitano la modalità PiP per ciascuno di essi (C = Click di `PictureInPictureButton`, R = Ridimensionamento della finestra del browser).

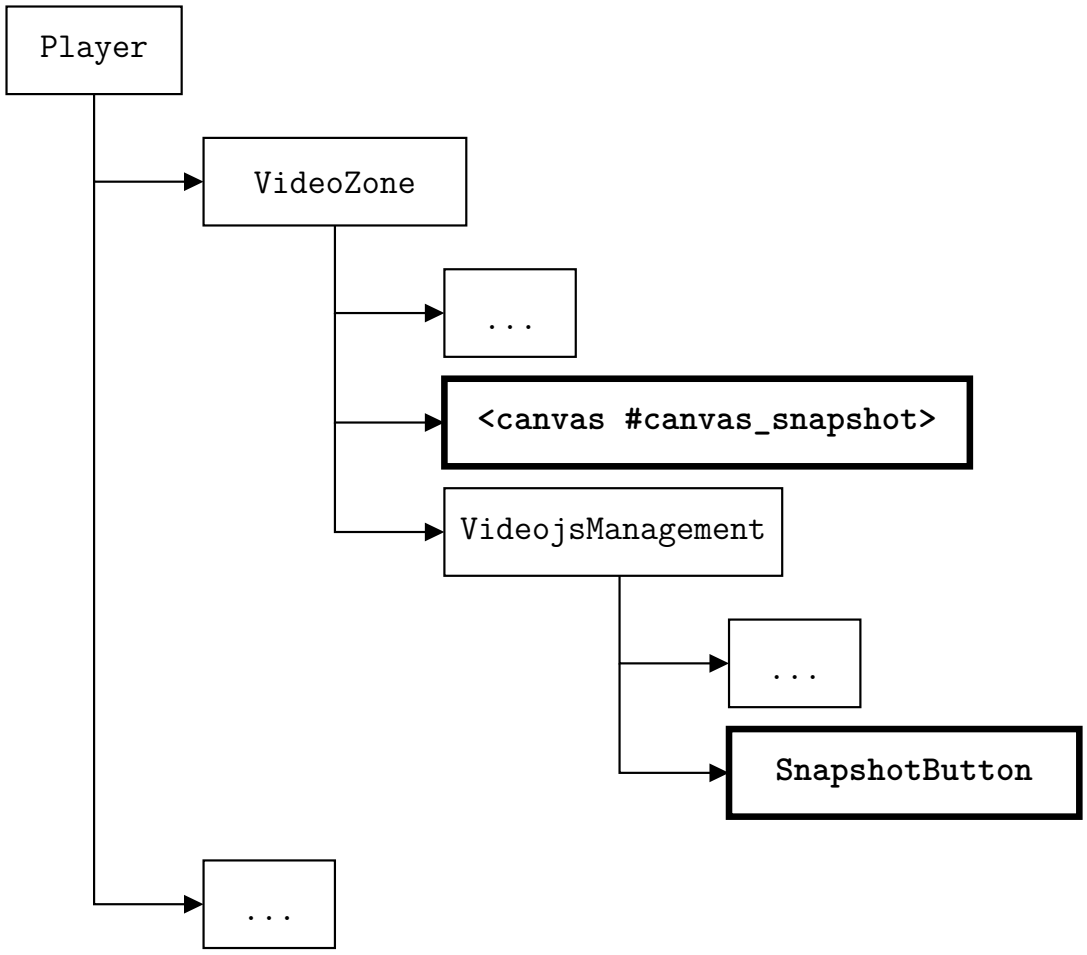

Figura 4.29: Gerarchia del player con l'aggiunta degli elementi e dei component per la cattura dei fotogrammi del video.

4.3 Risultati ottenuti

I contributi al codice di hls.js possono essere raccolti nel modulo logico `MyModule`: esso costituisce il nesso tra il contesto della libreria e il contesto esterno del player creato (figura 4.30), dato che racchiude sia le modifiche effettuate sui vari moduli di hls.js (evidenziati in grassetto in figura 4.31) che le aggiunte nei file di alto livello.

Il player aziendale, inizialmente indirizzato a contenuti esclusivamente audio, ora è un player audio/video/dati (A/V/D), (figura 4.32), in grado di riprodurre flussi A/V e mostrare le informazioni che viaggiano sul canale dati di HLS. La gerarchia dei component e degli elementi per il suo funzionamento è riportata in figura 4.33.

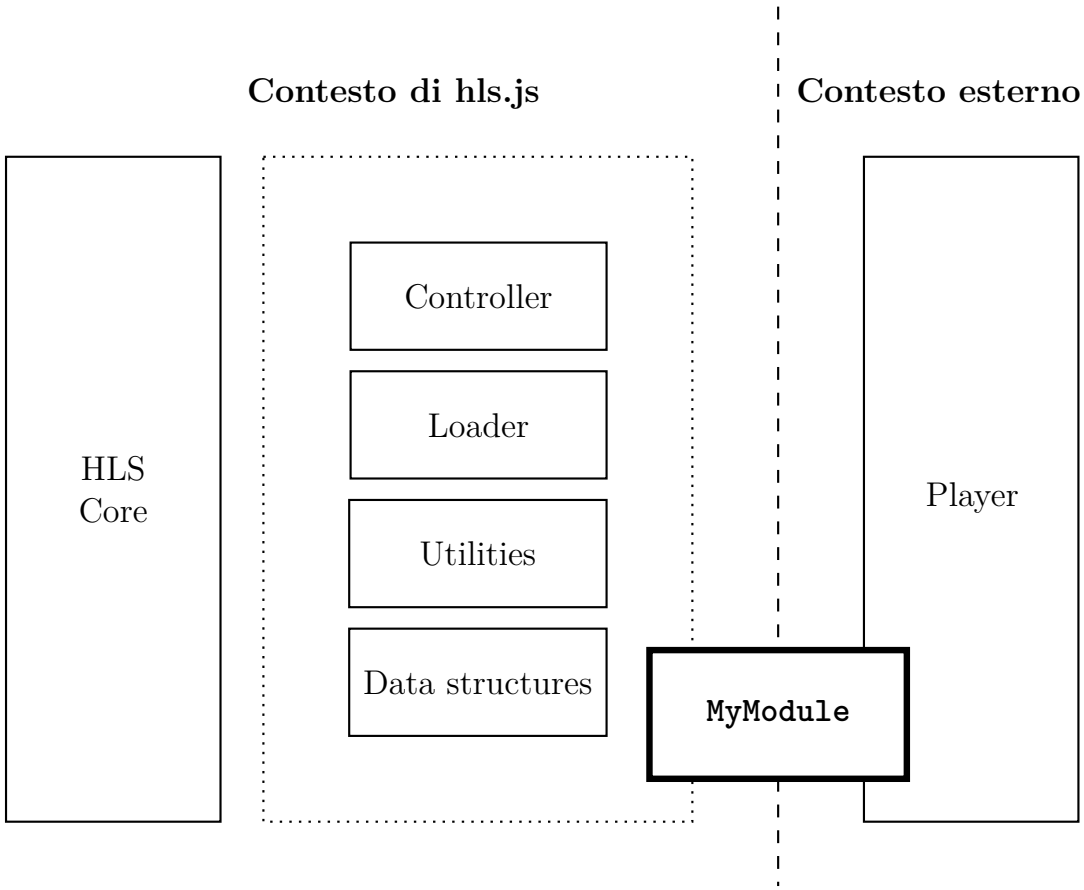

Figura 4.30: Introduzione di `MyModule` come ponte tra hls.js e il player.

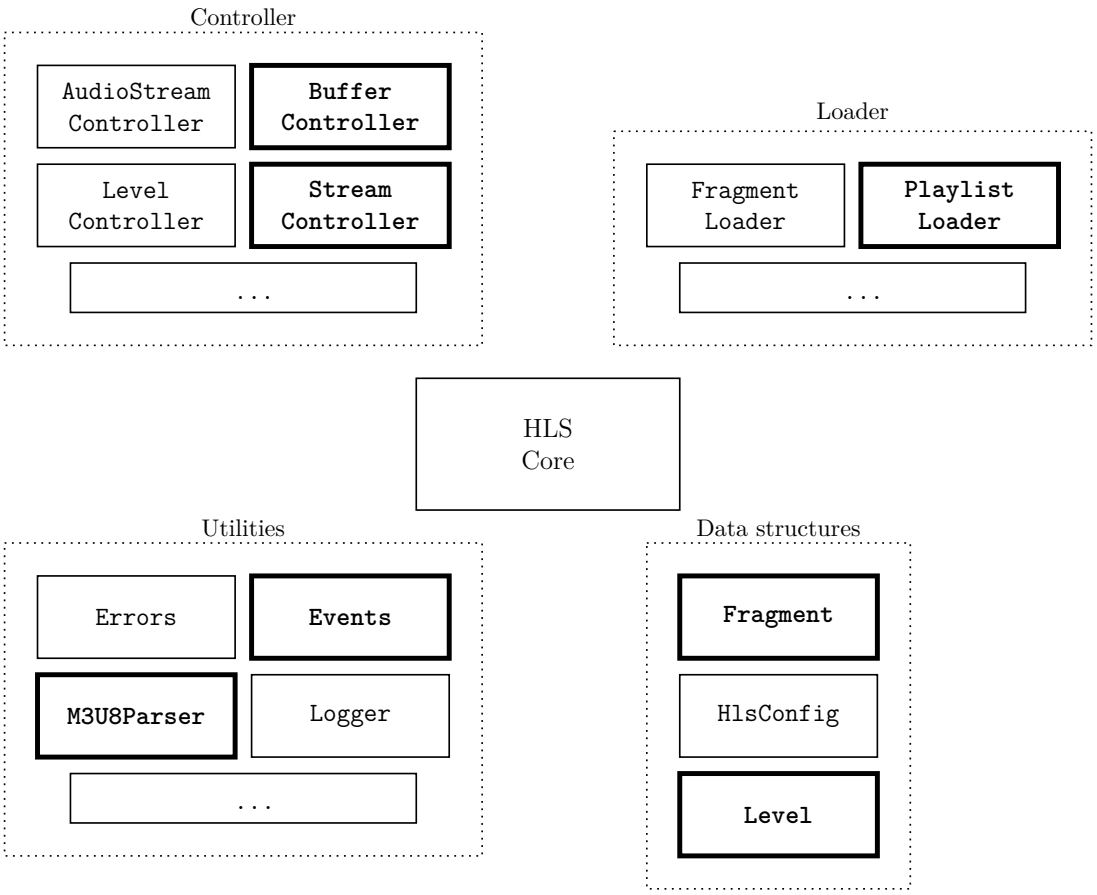

Figura 4.31: Architettura di hls.js con dettaglio dei moduli modificati.

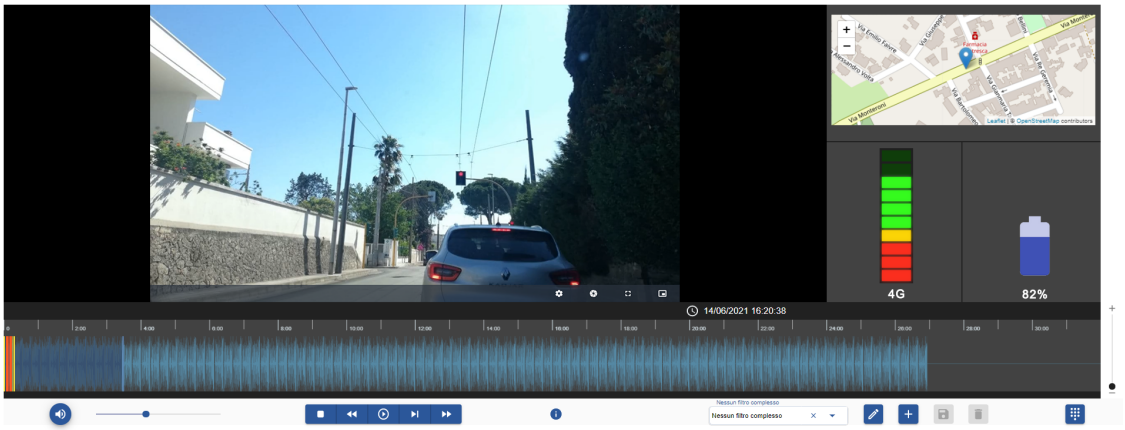

Figura 4.32: Il player A/V/D allo stato finale.

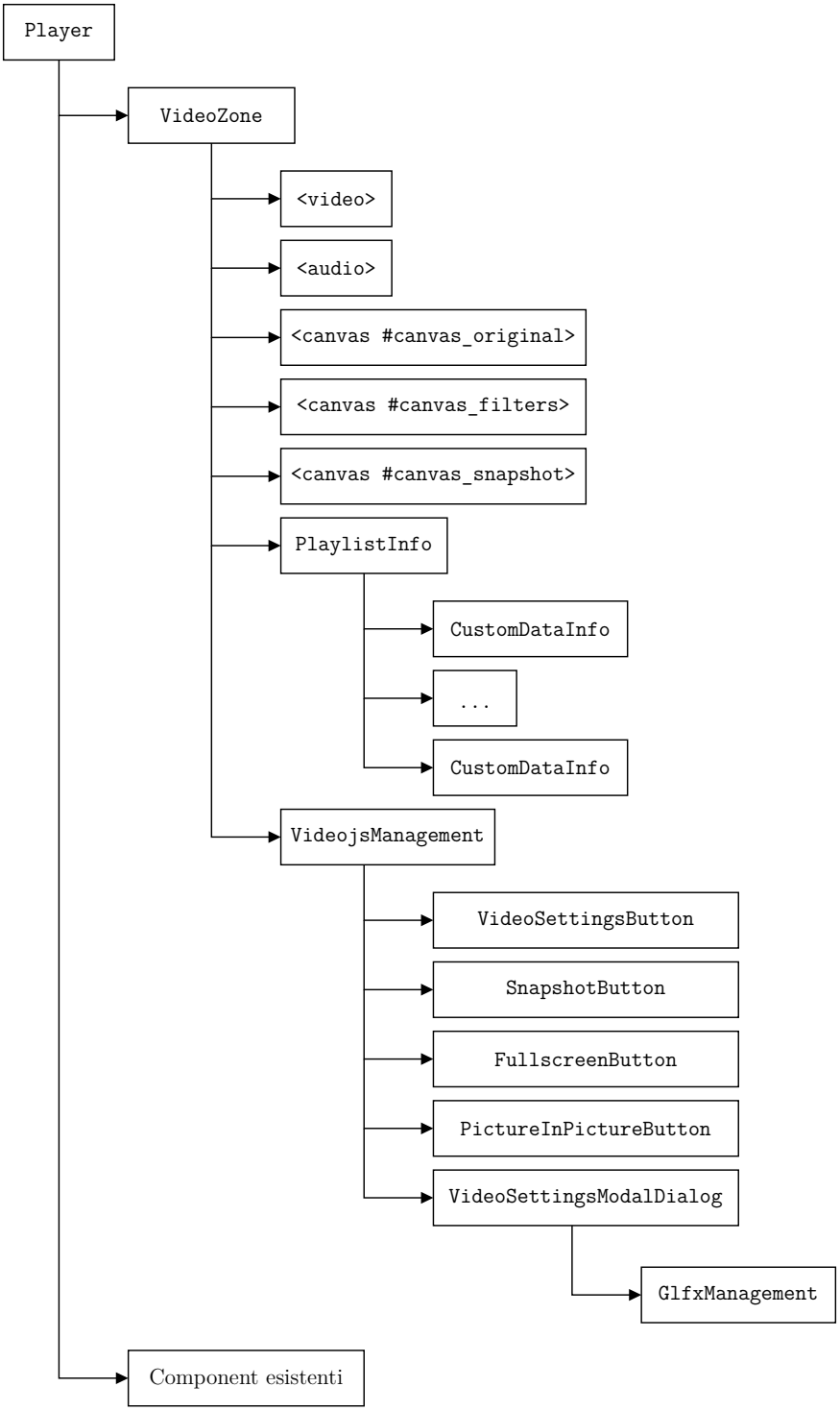

Figura 4.33: Gerarchia dei component e degli elementi del player A/V/D allo stato finale.

Capitolo 5

Sviluppi futuri e conclusioni

5.1 Object detection e recognition

5.1.1 Contesto e definizione

Nel corso degli ultimi anni sono stati compiuti numerosi passi in avanti nel campo della computer vision, una branca dell'Intelligenza Artificiale (IA) il cui obiettivo è la creazione di un modello del mondo reale tramite l'elaborazione digitale di immagini e video. Con l'evoluzione delle tecnologie al servizio della computer vision, questa scienza interdisciplinare si è diffusa in moltissime applicazioni anche di uso quotidiano: la guida autonoma di veicoli e robot, il face recognition per l'unlocking dello smartphone e la rilevazione di anomalie nell'ambito dell'elettronica biomedica sono solo alcuni esempi.

L'object detection e l'object recognition sono i principali ambiti della computer vision: il primo consiste nella localizzazione di oggetti in un'immagine, il secondo riguarda il riconoscimento degli oggetti e la capacità di ricondurli a classi predefinite.

Questi ambiti trovano una loro applicazione nel contesto della videosorveglianza, uno degli strumenti impiegati dagli addetti alle LI preventive per l'individuazione e la prevenzione di eventi come rapine, risse e attentati. L'object detection e recognition agevolano l'attività di videosorveglianza, poiché gli operatori, osservando

le immagini catturate dalle videocamere e contemporaneamente il prodotto della loro elaborazione da parte degli algoritmi, possono contattare tempestivamente le Forze dell'Ordine per un intervento sul luogo dove si sta compiendo - o si sta per compiere - l'attività illecita.

In considerazione di quanto appena descritto si considera, tra i possibili sviluppi futuri, una soluzione per l'applicazione di algoritmi di object detection e recognition sui fotogrammi del video e la visualizzazione dei risultati ottenuti all'interno del player.

5.1.2 La libreria face-api.js

Al momento face-api.js [40], una libreria JavaScript per il face detection e recognition all'interno di immagini e video, rappresenta un buon punto di partenza per le sperimentazioni iniziali. Essa si basa sul caricamento di appositi modelli forniti da TensorFlow, un'API open-source per il machine learning, e consente di rilevare l'area e i tratti del volto (sopracciglia, occhi, naso, bocca e forma del viso), lo stato emotivo del soggetto attraverso la mimica facciale nonché una stima dell'età e del genere della persona. Inoltre, face-api.js è in grado di confrontare i volti per fornire il grado di somiglianza tra i soggetti. Come si vede in figura 5.1 il risultato dell'analisi è mostrato su un elemento `<canvas>` che si sovrappone all'immagine o al video di partenza.

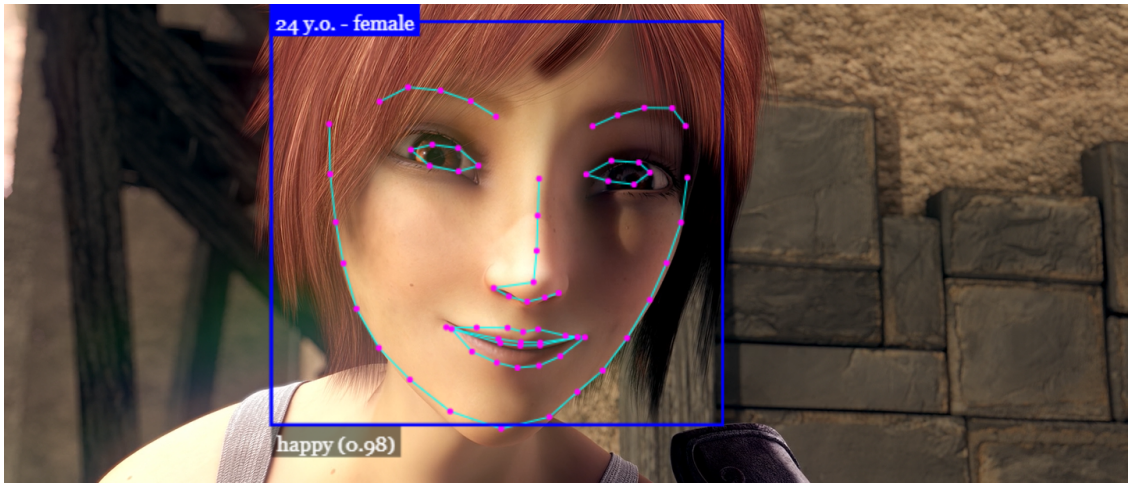

Figura 5.1: Esempio applicativo della libreria face-api.js.

Lo schema in figura 4.24 viene aggiornato con l'introduzione degli algoritmi di object detection e recognition (si veda la figura 5.2).

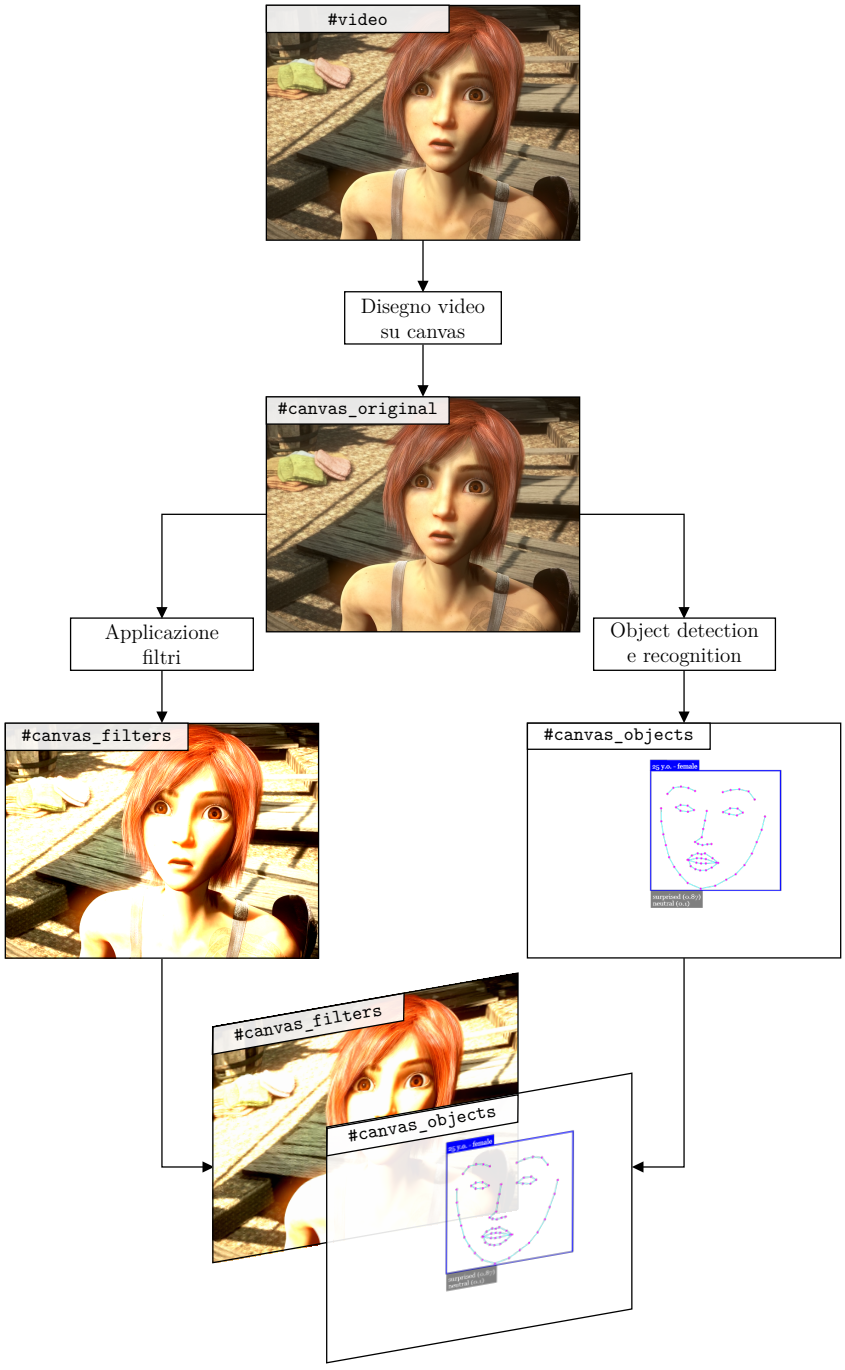

Figura 5.2: Schema del funzionamento di glfx.js con l'aggiunta degli algoritmi di object detection e recognition.

face-api.js, come suggerisce il nome, è dedicata al riconoscimento esclusivamente facciale, ma potrebbe essere espansa per implementare il riconoscimento dell'intera figura di una persona, di elementi specifici del corpo umano (come mani, braccia e gambe), di animali o di elementi utili nel contesto delle intercettazioni (ad esempio mezzi di trasporto e armi).

5.1.3 Possibile implementazione

Si consideri l'implementazione degli algoritmi di object detection e recognition lato client. Questa fase prevede i seguenti step:

1. caricamento dei modelli di TensorFlow;
2. aggancio all'elemento HTML sorgente (`<img>`, `<canvas>` o `<video>`);
3. chiamate ai metodi appositi.

L'implementazione lato client risulta vantaggiosa negli scenari d'uso caratterizzati, ad esempio, da ambienti poco illuminati, dove i fotogrammi devono necessariamente essere elaborati con l'applicazione di filtri video (ad esempio il contrasto e la luminosità) affinché la logica degli algoritmi di detection e recognition possa funzionare in maniere corretta. Si ottiene così un'architettura simile a quella descritta dallo schema in figura 5.3. Tuttavia, poiché la fase di caricamento dei modelli può rappresentare un'operazione dispendiosa in termini di risorse di sistema, è necessario valutare una soluzione alternativa.

Nel caso di implementazione lato server si potrebbe ricorrere al canale dati di HLS espandendo il set di metatag, adattando il parser e aggiungendo nuovi eventi e listener. Ad esempio, si definisce il metatag locale `#EXT-DETECTION` con la sintassi mostrata nel listato 5.1.

```
1 #EXTINF:4,
2 /* #EXT-DETECTION:{categoria,[caratteristiche]},{coordinate} */
3 #EXT-DETECTION:{face,[male,48,angry]},{[1.0,1.0],[2.0,2.0],...}
4 #EXT-DETECTION:{weapon,[knife]},{[3.0,3.0],[4.0,4.0],...}
5 segment00001.ts
```

Listato 5.1: Estratto di una media playlist contenente il metatag locale per l'object detection e recognition.

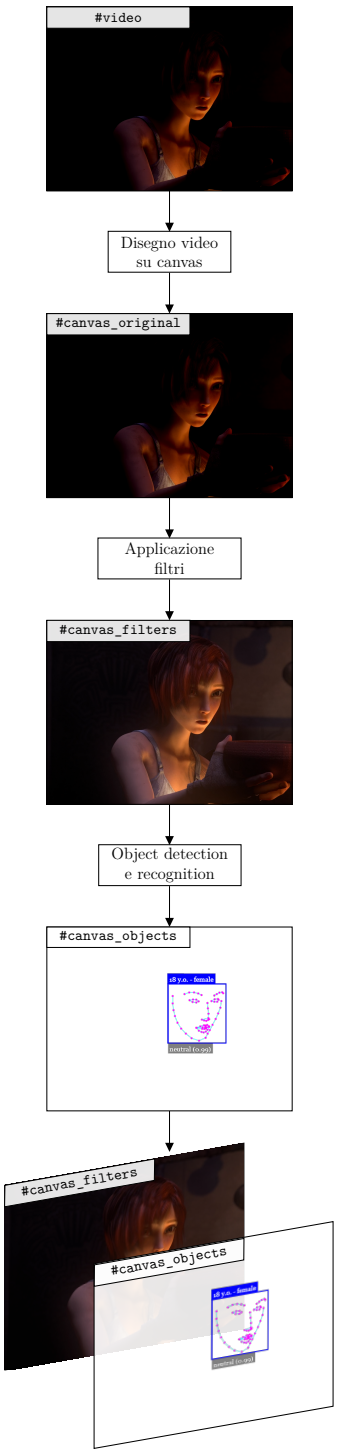

Figura 5.3: Schema alternativo del funzionamento di glfx.js con l'aggiunta degli algoritmi di object detection e recognition.

Si considerino i seguenti due parametri:

- la frequenza f con cui i fotogrammi all'interno dei chunk della playlist sono prelevati e analizzati dagli algoritmi di object detection e recognition;
- il numero n di oggetti rilevati in un determinato fotogramma del video.

La mole Q di byte da inviare al ricevitore attraverso il canale dati di HLS è direttamente proporzionale ad entrambi i parametri: perciò, l'applicazione web, a fronte di valori elevati di f e n, può subire un appesantimento e andare ineluttabilmente incontro a rallentamenti. Per prevenire tale evenienza si può tentare di diminuire il valore di f, di n o di entrambe le grandezze, ma non sempre quest'operazione rappresenta la soluzione al problema e, anzi, rischia di provocare effetti collaterali. A titolo esemplificativo, si consideri uno scenario realistico costituito da:

- una videocamera di sorveglianza V attiva puntata su una strada;
- un sistema di elaborazione E delle immagini che con una frequenza f estrae il fotogramma corrente dal video catturato da V e ne applica gli algoritmi di object detection e recognition;
- un'automobile A sospetta in transito.

Ad un certo punto della sua attività, la videocamera V riprende l'automobile A per un secondo. Se il sistema E lavora a una frequenza f nell'intervallo $2 \div 10$ frame/s, esso è in grado di prelevare ed elaborare un numero accettabile di fotogrammi salienti in cui compare il mezzo A. Tuttavia, se si diminuisce f fino a valori molto bassi (ad esempio $f = 0.5$ frame/s, cioè un fotogramma ogni due secondi) si ottiene senz'altro una riduzione di Q, ma è altamente probabile che si verifichi uno dei seguenti casi:

- nessun fotogramma contenente A passa al vaglio degli algoritmi di object detection e recognition;
- il sistema E estrae fotogrammi in cui il mezzo A è solo parzialmente visibile.

Dunque, in situazioni riconducibili allo scenario appena descritto è necessario decidere con attenzione il valore di f per stabilire un tradeoff tra le prestazioni generali del sistema E e la sua usabilità.

Il parametro n, poiché dipende dall'ambiente sorvegliato, non può essere controllato direttamente dagli operatori. Tuttavia, si può decidere, a seconda delle caratteristiche dell'ambiente e delle esigenze degli addetti alla videosorveglianza, di filtrare determinate categorie di oggetti (come ad esempio persone, autoveicoli, semafori etc.) e quindi applicare selettivamente gli algoritmi di object detection e recognition.

Un altro stratagemma per ridurre Q senza invalidare l'usabilità dell'applicazione potrebbe consistere nella creazione sul server di una playlist secondaria. Il server tiene traccia sia della playlist originale che di una playlist copia dell'originale ma ripulita da qualsiasi contenuto dati. Le informazioni per il canale dati sono registrate non solo nel file della playlist principale, ma anche in una tabella dedicata all'interno di un DB, ossia una collezione persistente e strutturata di dati. In questo modo il client:

- quando riproduce il contenuto multimediale in tempo reale, riceve la playlist originale dal server;
- ogni volta che consulta la registrazione A/V in differita non richiede i frammenti della playlist principale al server, bensì interroga il DB per ottenere, attraverso la generazione di eventi formali, i dati analizzati dagli algoritmi di object detection e recognition agli istanti consultati.

Alla luce di quanto descritto finora si evince che le scelte implementative in materia di object detection e recognition devono essere compiute sulla base di un'accurata fase di pianificazione e sperimentazione.

5.2 Considerazioni finali

Dopo aver inquadrato le Lawful Interception e, in particolare, le LI ambientali, si è voluto sottolineare l'importanza del tempo reale in questo contesto e nelle LI con finalità preventive. Si è posta dunque l'attenzione sugli ambienti a connettività di rete limitata e sulle problematiche che essi introducono nelle attività svolte dagli addetti alle intercettazioni.

Poiché il modello di live streaming precedentemente impiegato nelle intercettazioni ambientali si è rilevato non adatto a questo scopo, sono state condotte ricerche

approfondite sui principali formati container e protocolli adottati nella trasmissione di flussi multimediali: sfruttando i vantaggi del formato MPEG-TS e del protocollo HLS, è stato definito un nuovo modello resiliente alle disconnessioni di rete e quindi adatto alle LI ambientali in ambienti WiFi critici.

La libreria hls.js, impiegata nell'implementazione di HLS nel player multimediale, è stata analizzata adottando un approccio top-down: partendo dalle funzionalità di base e dai metodi di più alto livello, si è modificato il codice sorgente della libreria in maniera tale da raggiungere gli obiettivi prefissati. Inoltre, gli sviluppi su hls.js rappresentano un valido contributo al RFC 8216, che descrive il funzionamento di HLS.

Il player web sviluppato da RCS ETM Sicurezza Spa è stato arricchito con i canali video e dati di HLS, l'acquisizione dei fotogrammi, la possibilità di applicare filtri video e le modalità fullscreen e Picture-in-Picture. Adesso non è più un player solamente audio, bensì un player HLS audio/video/dati completo, in grado cioè di mostrare sia contenuti A/V che dati secondo le modalità definite dall'utente, e caratterizzato da una UI semplice e intuitiva.

Infine, l'applicazione web, grazie alla sua architettura modulare definita tramite i component di React, si apre a numerosi sviluppi futuri: il più utile nel contesto delle LI ambientali consiste senza dubbio nell'implementazione degli algoritmi di object detection e recognition per il riconoscimento sia facciale che di oggetti utili alle indagini. Questi algoritmi, se implementati in maniera opportuna, rappresenteranno un potente strumento al servizio degli investigatori per l'individuazione e la prevenzione di eventuali atti criminali.

Bibliografia

[1] *Lawful Interception (LI)*. URL: https://www.etsi.org/technologies/lawful-interception.

[2] Wikipedia. *Intercettazione — Wikipedia, L'enciclopedia libera*. [Online; in data 14-maggio-2021]. 2020. URL: https://it.wikipedia.org/w/index.php?title=Intercettazione&oldid=115427867.

[3] Wikipedia contributors. *Lawful interception — Wikipedia, The Free Encyclopedia*. [Online; accessed 7-May-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Lawful_interception&oldid=1018932656.

[4] *Lawful Interception (LI); Handover Interface and Service-Specific Details (SSD) for IP delivery; Part 1: Handover specification for IP delivery*. Technical Specification TS 102 232-1. Ver. 3.23.1. ETSI, mar. 2021, p. 11. URL: https://www.etsi.org/deliver/etsi_ts/102200_102299/10223201/03.23.01_60/ts_10223201v032301p.pdf.

[5] *Lawful Interception (LI); Retained data handling; Handover interface for the request and delivery of retained data*. Technical Specification TS 102 657. Ver. 1.27.1. ETSI, apr. 2021. URL: https://www.etsi.org/deliver/etsi_ts/102600_102699/102657/01.27.01_60/ts_102657v012701p.pdf.

[6] Angela Allegria. *Le intercettazioni preventive*. Nov. 2016. URL: https://www.diritto.it/le-intercettazioni-preventive/.

[7] Traci Ruether. *The Complete Guide to Live Streaming*. Nov. 2019. URL: https://www.wowza.com/blog/complete-guide-to-live-streaming.

[8] *Information technology — Generic coding of moving pictures and associated audio information — Part 1: Systems*. Standard ISO/IEC 13818-1:2019. ISO, giu. 2019, p. 11. URL: https://www.iso.org/obp/ui/#iso:std:75928:en.

[9] Wikipedia contributors. *MPEG transport stream — Wikipedia, The Free Encyclopedia*. [Online; accessed 11-May-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=MPEG_transport_stream&oldid=1006751754.

[10] Max Wilbert. *What is HLS Streaming and When Should You Use It? [2021 Update]*. Apr. 2021. URL: https://www.dacast.com/blog/hls-streaming-protocol/.

[11] Wikipedia contributors. *HTTP Live Streaming — Wikipedia, The Free Encyclopedia*. [Online; accessed 13-May-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=HTTP_Live_Streaming&oldid=1018384038.

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

[13] *HTTP Live Streaming - An Emerging Standard in Adaptive Bitrate Video*. URL: https://www.encoding.com/http-live-streaming-hls/.

[14] Per Frojdh, David Singer e Randall Gellens. *The 'Codecs' and 'Profiles' Parameters for "Bucket" Media Types*. RFC 6381. Ago. 2011. DOI: 10.17487/RFC6381. URL: https://rfc-editor.org/rfc/rfc6381.txt.

[15] Max Wilbert. *HTTP Live Streaming (HLS) – The Pros and Cons and How it Works*. Apr. 2021. URL: https://www.dacast.com/blog/http-live-streaming/.

[16] Uros Stanimirovic. *The Ins and Outs of HTTP Live Streaming Protocol (HLS)*. Set. 2020. URL: https://www.brid.tv/hls-streaming/.

[17] Kevin Graham. *HLS Encryption: How to Encrypt Videos in AES-128 For HTTP Live Streaming [2021 Update]*. Apr. 2021. URL: https://www.dacast.com/blog/hls-encryption-for-video/.

[18] Tim Siglin. *HLS Now Supports Fragmented MP4, Making it Compatible With DASH*. Giu. 2016. URL: https://www.streamingmedia.com/Articles/ReadArticle.aspx?ArticleID=111796.

[19] Phil Cluff. *The Low Latency Live Streaming Landscape in 2019*. Gen. 2019. URL: https://mux.com/blog/the-low-latency-live-streaming-landscape-in-2019/.

[20] Traci Ruether. *Update: What is Low-Latency HLS and How Does It Relate to CMAF*. Feb. 2020. URL: https://www.wowza.com/blog/apple-low-latency-hls.

[21] Traci Ruether. *IETF Incorporates Low-Latency HLS Into the HLS Spec*. Apr. 2020. URL: https://www.wowza.com/blog/ietf-incorporates-low-latency-hls-into-the-hls-spec.

[22] Shachar Zohar. *HTTP Live Streaming In Javascript*. Nov. 2015. URL: https://blog.peer5.com/http-live-streaming-in-javascript/.

[23] Mag. 2021. URL: https://github.com/video-dev/hls.js/blob/master/README.md.

[24] Mar. 2021. URL: https://github.com/video-dev/hls.js/blob/master/docs/design.md.

[25] MDN contributors. Mar. 2021. URL: https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement.

[26] MDN contributors. Mag. 2021. URL: https://developer.mozilla.org/en-US/docs/Web/API/HTMLMediaElement.

[27] MDN contributors. Apr. 2021. URL: https://developer.mozilla.org/en-US/docs/Web/API/HTMLVideoElement.

[28] MDN contributors. Apr. 2021. URL: https://developer.mozilla.org/en-US/docs/Web/API/HTMLAudioElement.

[29] MDN contributors. Feb. 2021. URL: https://developer.mozilla.org/en-US/docs/Web/API/MediaSource.

[30] MDN contributors. Feb. 2021. URL: https://developer.mozilla.org/en-US/docs/Web/API/SourceBuffer.

[31] MDN contributors. Mag. 2021. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer.

[32] URL: https://reactjs.org/.

[33] URL: https://angular.io/.

[34] MDN contributors. Giu. 2021. URL: https://developer.mozilla.org/en-US/docs/Web/Web_Components.

[35] MDN contributors. Mar. 2021. URL: https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_shadow_DOM.

[36] URL: https://wavesurfer-js.org/.

[37] URL: https://leafletjs.com/.

[38] URL: https://videojs.com/.

[39] URL: https://evanw.github.io/glfx.js/.

[40] URL: https://github.com/justadudewhohacks/face-api.js/.