Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Dynamic Adaptive Streaming over Http: implementazione e analisi sperimentale di algoritmi di rate adaptation su client Android

Relatore:
Chiar.mo Prof.
Luciano Bononi

Correlatore:
Dott. Luca Bedogni

Presentata da:
Giuseppe Carnà

Sessione I
Anno Accademico 2014/2015

"The only way to do great work is to love what you do. If you haven't found it yet, keep looking. Don't settle." - Steve Jobs

Abstract

Il lavoro svolto in questa tesi consiste nell'implementare un'applicazione Android per lo streaming video, conforme allo standard MPEG-DASH. L'obiettivo è quello di fornire un valido strumento al fine di eseguire delle analisi sperimentali su algoritmi particolari, detti di rate adaptation. MPEG-Dynamic Adaptive Streaming over Http è uno standard emergente ed è considerato da molti il futuro dello streaming multimediale. Questa tecnologia consente di auto-regolare la qualità del video in base alle condizioni della rete, la capacità del dispositivo o le preferenze dell'utente. Inoltre, essendo uno standard, permette di rendere interoperabili i server e i device dei vari fornitori di contenuti multimediali. Nei primi capitoli introduttivi verrà presentato lo standard e i lavori correlati, successivamente verrà descritta la mia proposta applicativa: DashPlayer. In conclusione, verrà compiuta una valutazione sperimentali sugli algoritmi sopracitati che costituiscono la parte logico-funzionale dell'applicazione.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

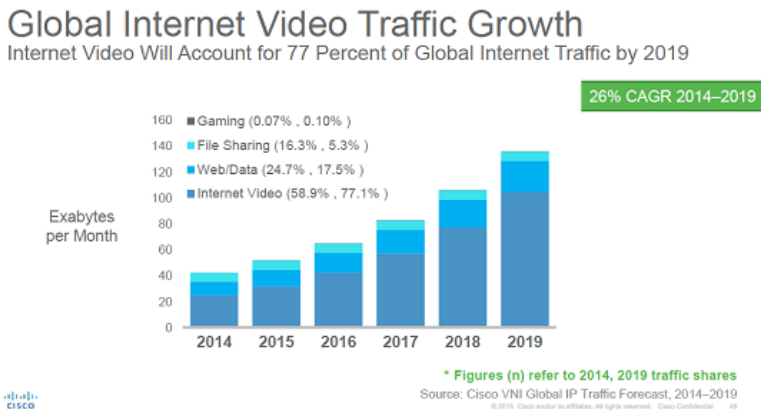


Figura 1.1: Crescita del traffico globale di internet fino al 2019

1.1 Scenari e obiettivi

La diffusione e lo sviluppo delle tecnologie informatiche a cui abbiamo assistito negli ultimi anni, hanno fatto si che termini nel campo delle telecomunicazioni o dell'informatica, come "Streaming", "Buffering" o "Codec" non necessitino di presentazioni particolari anche ai meno esperti del settore. Siamo spettatori non paganti di quella che continuerà ad essere la crescita di Internet e oggi i dati multimediali (audio/video) occupano in percentuale la

maggior parte dei contenuti presenti sulla rete [1]. Questa continua crescita ha reso indispensabile il miglioramento delle prestazioni di tutte le tecniche utilizzate per la loro distribuzione. In particolare, nell' erogazione di materiale audio/video in tempo reale, ovvero quello che è comunemente chiamato Streaming, necessita l'utilizzo di tecniche particolari per poter garantire una buona *Quality Of Service* grazie agli strumenti della scienza Informatica e alle tecnologie ad essa annesse. Dal grafico in fig 1.1 possiamo osservare quanto affermato precedentemente: i contenuti video costituiscono la maggior parte del traffico globale di Internet e continueranno a crescere superando il 70% entro il 2019 [1]. La verità è che lo streaming multimediale è ancora alla sua infanzia comparato alla sua potenza commerciale, eppure viste le prospettive di crescita ancora oggi esistono una miriade di protocolli, tecnologie e tecniche di streaming diverse. Ogni piattaforma commerciale è un sistema chiuso con il proprio manifest file, il proprio formats content ed il proprio protocollo [14]. Detto in altri termini, non esiste compatibilità tra i device e i server dei diversi fornitori e questo non favorisce la crescita nel settore. Proprio per queste ragioni si sentì il bisogno di realizzare uno standard per rendere interoperabili i vari device e server. L'interoperabilità è fondamentale per la crescita del mercato multimediale, poiché solo un ecosistema comune di contenuti e servizi, indipendenti da piattaforme e da tecnologie proprietarie, sarà in grado di soddisfare la crescente richiesta da parte di un' ampia varietà di dispositivi presenti sul mercato, come PC, TV, laptop, game console, tablet e smartphone.

MPEG-DASH o Dynamic Streaming Over Http, chiamato per brevità anche DASH, è uno standard sviluppato anche per questo. Esso si basa su *Dynamic Adaptive Streaming*, ovvero una tecnica che permette di cambiare dinamicamente la qualità del video durante la riproduzione cercando di fornire la migliore *Quality Of Experience* dipendente dalle richieste e dai bisogni dell'utente. Per esempio, in presenza di una connessione di rete poco stabile potranno essere utilizzati degli specifici algoritmi di rate adaptation che monitorino la qualità della connessione e, quindi, decidano di tenere bassa la

qualità del video riducendo le possibili interruzioni del playback.

Uno degli scopi principali di questa tesi è stato quello di implementare un client Android conforme allo standard MPEG-DASH che permetta la trasmissione in streaming di contenuti multimediali utilizzando la tecnica di Dynamic Adaptive Streaming. Successivamente, si è cercato di ideare e implementare qualche algoritmo di rate adaptation riutilizzando alcuni già esistenti in letteratura per poter compiere un'analisi sperimentale decidendo quale sia l'algoritmo più adatto a vari scenari e casi d'uso.

1.2 Streaming: the state of the art

Il termine anglosassone *Streaming* è ormai divenuto un cliché diffuso nella società "internettiana" per riferirsi a qualsiasi forma di distribuzione multimediale (audio/video) sia lecita che illecita. Possiamo definire con il termine streaming tutto ciò che identifica un flusso di dati audio/video trasmessi da una sorgente ad una o più destinazioni tramite una rete, ma occorre tuttavia fare una distinzione in base alle tecniche utilizzate.

In questo capitolo verranno introdotti le varie tipologie, protocolli e tecnologie utilizzati in questo ambito.

1.3 Tipologie di streaming

Essenzialmente abbiamo tre tipi di diffusione telematica dei contenuti multimediali [5]:

Download and play È il metodo classico per la distribuzione multimediale, dove il file deve essere scaricato completamente prima di poter essere riprodotto.

Progressive download Permette la riproduzione del file durante la fase di download; lo svantaggio è che, se non si dispone di una connessione abbastanza veloce, la riproduzione potrebbe subire delle interruzioni.

True streaming Non salva nessun file sul disco fisso dell'utente, la riproduzione del video avviene in real-time.

Inoltre in base al tipo di contenuti queste tre categorie possono essere ancora generalizzate:

Streaming on demand I contenuti sono codificati e risiedono comunemente su un server e sono inviati al client dopo averne fatto richiesta. Non sarà necessario scaricare tutto per intero per iniziare la riproduzione: i dati ricevuti vengono decompressi dal codec e riprodotti pochi secondi dopo l'inizio della ricezione.

Streaming live Simile alla tradizionale trasmissione radio o video in broadcast. Anche in questo caso i dati sono trasmessi utilizzando opportune compressioni per alleggerire il più possibile il carico di lavoro sulla rete. La compressione dei contenuti introduce nel flusso un ritardo di pochi secondi. Sebbene a volte per trasmissioni via web si utilizzi il termine streaming come sinonimo di "diretta" è da chiarire che un contenuto in streaming può essere live oppure no [2].

1.4 Protocolli e Standard

In questa sezione introdurrò un po' di terminologia e spiegherò, senza dettagliare troppo, alcune sigle che, sicuramente, tutti i più incalliti "internauti" avranno incontrato almeno una volta nella vita durante le loro sessioni di web streaming.

Definizione di standard

Uno *Standard* è un insieme di norme e caratteristiche che un sistema, prodotto, o qualsiasi altro servizio da realizzare deve soddisfare a livello logico-funzionale [6]. Uno standard, quindi, garantisce l'interoperabilità tra il software, lasciando la libertà di realizzare il software in mano ai produttori.

Solitamente, uno standard, prima di essere considerato tale, viene analizzato dai vari enti internazionali presenti sul globo.

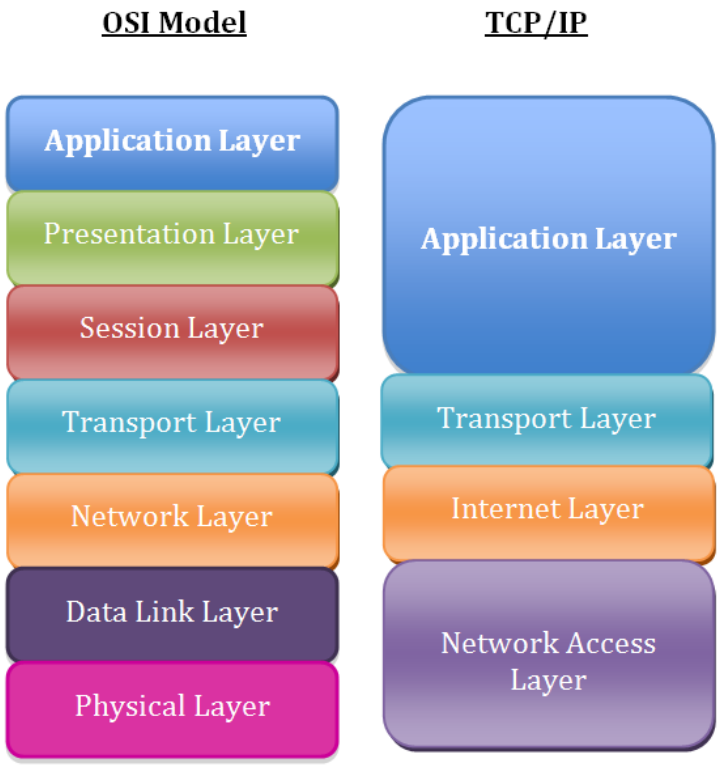


Figura 1.2: confronto fra modello ISO/OSI (standard de jure) e modello TCP/IP (standard de facto)

Gli standard possono essere categorizzati in:

Standard de jure standard approvato da un ente internazionale senza che sia di applicazione obbligatoria. Per esempio Il modello ISO/OSI rappresenta uno standard de jure [6].

Standard de facto standard sviluppato da una o più aziende e che ha acquisito importanza generale grazie alla sua diffusione del mercato [6]. Il modello TCP/IP è un esempio di standard de facto.

Organizzazioni di Standardizzazione

Infine, una piccola descrizione di alcuni enti di standardizzazione più famosi:

ISO (International Standards Organisation) È la più importante organizzazione a livello mondiale per la definizione di norme tecniche.

ITU-T (International Telecommunications Union-Telecommunication Standards Sector) Standard per le telecomunicazioni, telefonia e trasferimento dati

ANSI (American National Standards Institute) Organizzazione privata per l'unificazione degli standard in USA

IEEE (Institute of Electrical and Electronics Engineers) È un'associazione internazionale di scienziati professionisti con l'obiettivo della promozione delle scienze tecnologiche.

Definizione di protocollo

È un insieme di regole formalmente descritte, definite al fine di favorire la comunicazione tra una o più entità [7]. Tutte queste regole sono definite mediante specifici protocolli, dalle tipologie più varie e ciascuno con precisi compiti/finalità, a seconda delle entità interessate e il mezzo di comunicazione. Per scambiarsi dati due agenti devono attenersi ad un protocollo predefinito e ben specificato. Di seguito una breve descrizione di alcuni dei protocolli più utilizzati:

TCP È un protocollo di rete a pacchetto di livello di trasporto: si occupa di controllo di trasmissione ovvero rendere affidabile la comunicazione dati in rete tra mittente e destinatario.

UDP È un protocollo di livello di trasporto a pacchetto, usato di solito in combinazione con il protocollo di livello di rete IP.

IP È un protocollo di rete a pacchetto.

Protocolli di livello applicazione utilizzati in ambito streaming

Nel momento in cui selezioniamo il tasto play del nostro abituale sito di streaming ha inizio la comunicazione fra il nostro client e il server web dove risiede il video, attraversando tutta la pila TCP/IP. Questo significa che ogni livello risolve una serie di problemi nella trasmissione dei dati e fornisce uno specifico servizio ad un livello più alto. Ogni livello più alto si avvicina sempre di più all'interfaccia utente e poiché il mio lavoro di tesi verte, soprattuto, a fornire un servizio al livello applicazione tralascerò volutamente di descrivere nel dettaglio i livelli più bassi dello stack TCP/IP.

RTP Real-time Transport Protocol è un protocollo del livello applicazione utilizzato per servizi di comunicazione in tempo reale su Internet ed è basato, soprattutto, sul protocollo UDP al livello di trasporto.

RTCP Real-time Transport Control Protocol è un protocollo che raccoglie statistiche sulla qualità del servizio del protocollo RTP e trasporta le informazioni riguardo ai partecipanti ad una sessione. Può utilizzare diversi protocolli del livello di trasporto, incluso Transmission Control Protocol (TCP) o Universal Datagram Protocol (UDP). Si tratta di un protocollo stateful: una volta che è stata stabilità la connessione con il server, quest'ultimo tiene traccia dello stato del client fintanto che esso rimane collegato. Il server invia il file multimediale come un continuo stream di pacchetti utilizzando il livello transport [11].

HTTP Hypertext Transfer Protocol è il protocollo più diffuso per la trasmissione di dati sul web [7]. La logica del HTTP è basata sul concetto di ricevente/servente (client/server): il client manda una richiesta ed il server ritorna una risposta. Le specifiche del protocollo sono gestite dal World Wide Web Consortium (W3C) [7]. Al contrario del protocollo RTCP, l'HTTP è un protocollo stateless, se un client HTTP fa richiesta di un dato, il server risponde inviando il dato richiesto e conclude terminando la transazione; utilizza il protocollo TCP al livello di trasporto.

Oggi, lo streaming con HTTP è il più diffuso ed è diventato estremamente popolare negli ultimi anni, sebbene precedentemente, gli studi effettuati su video streaming raccomandavano l'utilizzo del protocollo RTP utilizzando l'User Datagram Protocol a livello di trasporto, poiché sostenevano che non si sarebbero potuti trasportare in modo soddisfacente i dati multimediali tramite il TCP, utilizzato da HTTP al livello di trasporto, principalmente a causa dei ritardi dovuti alla ritrasmissione dei pacchetti e alla sua ampia variazione del Throughput [11]. Differentemente da RTP, HTTP è *firewalll friendly* e può avvalersi di alcuni componenti presenti nella struttura di Internet odierna come proxy, cache e CDN.

1.5 Dynamic Adaptive Streaming

Dynamic Adaptive Streaming è una tecnica particolare, basata su HTTP, che adegua la qualità dello streaming video alle risorse disponibili nel dispositivo dell'utente, quali le condizioni della rete (*bandwith*) o la capacità della CPU, avendo come risultato il miglioramento della QoE dell'utente.

> "The result: very little buffering, fast start time and a good experience for both high-end and low-end connections" cit.[10]

1.5.1 MPEG-DASH

Dynamic Adaptive streaming over HTTP, anche conosciuto come MPEG-DASH, o semplicemente DASH, è stata la prima soluzione Dynamic Adaptive Streaming ad essere riconosciuta come uno standard internazionale. Il funzionamento è abbastanza semplice: la risorsa multimediale viene partizionata in uno o più segmenti e spedita al client tramite il protocollo HTTP. Prima di ricevere i segmenti, il client riceve dal server un file MPD (Media Presentation Description), che descrive le informazioni sui segmenti. Dopo aver effettuato un parsing sul MPD, il client acquisisce tutte le informazioni utili per effettuare lo streaming e basandosi su vari algoritmi di rate adaptation, cerca di

adattare la qualità dello streaming alle condizioni della rete. Lo standard MPEG-DASH fornisce solo le specifiche su MPD e sul formato dei segmenti. Tutto il resto rimane fuori dallo *scope* di MPEG-DASH [15] lasciando libertà agli sviluppatori delle varie applicazioni, ma contemporaneamente, instaurando un'interoperabilità fra i vari server e device, che rimangono fedeli allo standard. I lavori su DASH iniziarono nel 2010 e divenne uno standard internazionale nel Novembre 2011. Lo standard fu pubblicato come ISO/IEC 23009-1:2012 in Aprile 2012 [13]. La Dash Industry Forum ne promosse l'adozione e aiutò la transizione da standard in un vero e proprio business. Essa è composta dai più grandi leader del settore, come Microsoft, Netflix, Google, Ericsson, Samsung, Adobe, ecc. ed ha come obiettivo quello di creare le linee guide del Dash nei vari casi d'uso.

1.6 Tecniche e algoritmi di compressione

Data l'enorme quantità di materiale multimediale presente sulla rete, la "compressione" dei dati riveste un ruolo fondamentale per garantire un ottimo livello di servizio da parte di tutte le piattaforme di video sharing esistenti. Possiamo utilizzare il termine inglese "codec" per indicare tutte quelle tecnologie usate per la compressione di dati.

Essenzialmente, esistono due tipi di codec:

Lossless codec Dopo la decodifica il file codificato viene riportato esattamente al suo stato originario.

Lossy codec Al contrario di lossless produce una perdita di informazione durante la codifica e il file dopo la decodifica non sarà mai riportato allo stato originale.

1.6.1 Algoritmi di compressione audio

Tra i più importanti [9] abbiamo:

- MP3 è un algoritmo di compressione lossy in grado di ridurre la quantità dei dati richiesti per memorizzare un suono mantenendo una qualità audio accettabile.
- Vorbis è un algoritmo libero per la compressione di tipo audio digitale lossy che permette di avere una maggiore compressione rispetto a MP3 utilizzando delle tecniche avanzate di psicoacustica [9].
- AAC è designato per essere il successore dell' MP3, fornendo una qualità audio superiore a quest'ultimo a parità di fattore di compressione.

1.6.2 Algoritmi di compressione video

Ne descrivo due dei più importanti [9]:

- H.264, spesso chiamato anche "Advanced Video Coding (AVC), è un formato standard di compressione video digitale lossy creato dal Moving Picture Experts.
- VP8 è l'ultimo codec video di On2 Technologies acquisita da Google: secondo dei test svolti da [8] si è concluso che H.264 ha un leggero vantaggio in termini di qualità, ma non rilevanti dal punto di vista commerciale.

Capitolo 2

Related Works

MPEG-Dynamic Adaptive Streaming over HTTP non è una novità per quanto riguarda Dynamic Adaptive Streaming. In questo capitolo metterò a confronto alcuni dei più famosi sistemi proprietari.

2.1 Sistemi proprietari

Attualmente, i più noti sistemi proprietari sono:

1. Apple's HTTP Live Streaming (HLS)
2. Microsoft's Smooth Streaming (MSS)
3. Adobes HTTP-based Dynamic Streaming (ADS)

Tutti questi protocolli presentano alcune similitudini, ma si differenziano per alcune caratteristiche: HLS lavora con l'estensione MPEG-2 Transport Stream (M2TS), mentre, Smooth Streaming e HDS ne usano altre diverse tra di loro.

Nel Novembre del 2011, Will Law Akamai, capo architetto di Akamai Technologies, commentò:

> "We've spent the past five years delivering a variety of adaptive video formats - SmoothHD, HDNI, HLS and HDS - all of which are 80 percent the same but 100 percent incompatible".

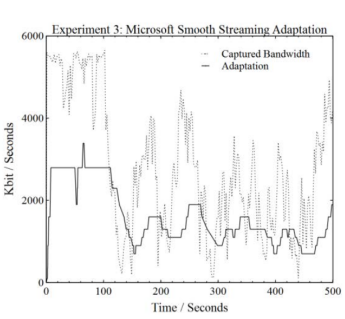
Questo significa che lo scopo di questi sistemi è, concettualmente, quello di arrivare a fornire lo stesso tipo di servizio, ovvero lo streaming "adattivo", ma utilizzando protocolli e standard diversi. Gi autori dell'articolo [11] hanno effettuato vari test ed analisi sulle tecniche di dynamic adaptive streaming citate: per esempio, in figura 2.1, osserviamo il comportamento di MSS, in particolare, 2.1a mostra come il bitrate del video si adatta con il throughput stimato, mentre in fig 2.1b, il livello del buffer mantiene un livello massimo vicino ai 30 secondi. MSS aumenta la qualità in modo progressivo (*stepwise manner*) e, dai test effettuati, si è dimostrato essere molto conservativo [11]. Questo è un risultato non negativo se si considera uno scenario con numerose fluttuazioni del throughput.

Sempre in [11] possiamo trovare dei test effettuati su ADS fig.2.3, questa volta contrariamente a MSS sembra essere molto più "aggressivo" nell'aumentare la qualità. Infatti, effettua il più delle volte degli switch tra la più alta e la più bassa representation reagendo poco bene alla variazioni del throughput e provocando dei repentini cali del buffer.

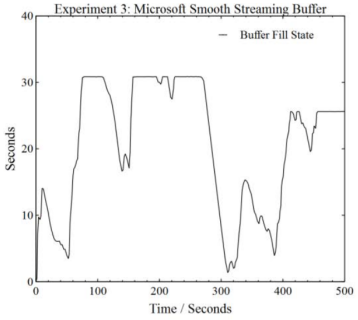
Infine, vediamo un'analisi effettuata sul client HLS di Apple, designato specialmente per scenari di mobilità: in figura 2.2a possiamo osservare che anch'esso reagisce ai cali del throughput in stepwise manner. Quando il buffer raggiunge il livello massimo di 200 secondi invia una richiesta di 75 segmenti in una sola volta provocandone un calo(oss. figura 2.2b dai secondi 100 a 200). Ciò accade perché non considera le fluttuazioni del throughput durante questi periodi di richieste *bundle* [11].

2.2 Risultati ottenuti dai test

I test sono stati effettuati in mobilità sull'autostrada di Carinthia/Austria; dalle analisi effettuate in [11], Microsoft Smooth Streaming si è dimo-

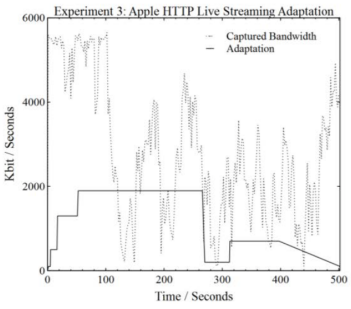


(a) MSS adaptation

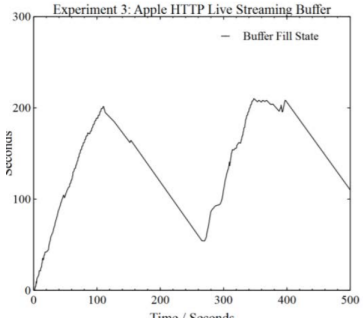


(b) MSS buffer

Figura 2.1: Analisi di Microsoft's Smooth Streaming

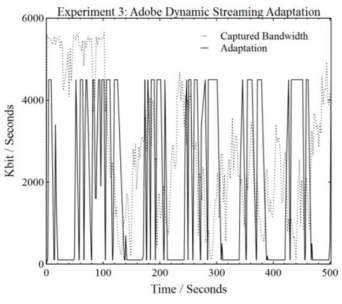


(a) HLS adaptation

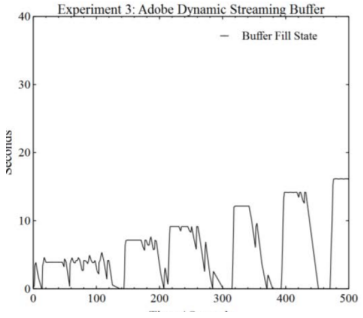


(b) HLS buffer

Figura 2.2: Analisi di Apple's HTTP Live Streaming



(a) ADS adaptation



(b) ADS buffer

Figura 2.3: Analisi di Adobes HTTP-based Dynamic Streaming

strato abbastanza prestante in questo scenario ottenendo il più alto average bitrate, ossia la qualità media più elevata. Invece, Apple Live Http Streaming, designato specialmente per la trasmissione video di Iphone e Ipad, è risulato essere troppo conservativo avendo riportando il più basso average bitrate. Infine, Adobe's Dynamic Adaptive Streaming si è dimostrato essere l'unico sistema a non avere uno *smooth playback* (playback intelligente) a causa dei continui stalli e re-buffering che offrono una scarsa QoE all'utente.

Capitolo 3

L'applicazione DashPlayer

Uno degli scopi della tesi è stato quello di progettare un'applicazione conforme allo standard MPEG-DASH, sul sistema operativo Android. In questo capitolo illustrerò le caratteristiche dell'architettura DASH e descriverò le caratteristiche principali dell'applicazione.

3.1 Dynamic Adaptive Streaming

Vediamo una breve panoramica del funzionamento di un sistema Dynamic Adaptive Streaming senza tenere conto delle specifiche dello standard MPEG: in figura 3.1 il contenuto multimediale è diviso in componenti video e audio; la risorsa video è codificata a tre differenti bitrate: 5 Mbps, 2 Mbps, 500 kbps. Il contenuto audio è disponibile in due lingue: audio 2 per la versione originale francese mentre audio 1 per quella doppiata in inglese, entrambi codificati AAC a 128 e 48 kbps. Il device inizia lo streaming richiedendo il segmento del video con il più alto bitrate (5Mbps) e con audio doppiato in lingua inglese. I numerini in figura indicano l'ordine cronologico di download dei segmenti, infatti dopo il download del primo segmento, il client si rende conto per un qualche motivo di non riuscire a supportare quel bitrate e decide di effettuare uno switch down a 2 Mbps rimanendo sempre con lo stesso segmento audio. Successivamente, effettua degli ulteriori switch down al punto 3 sia per il

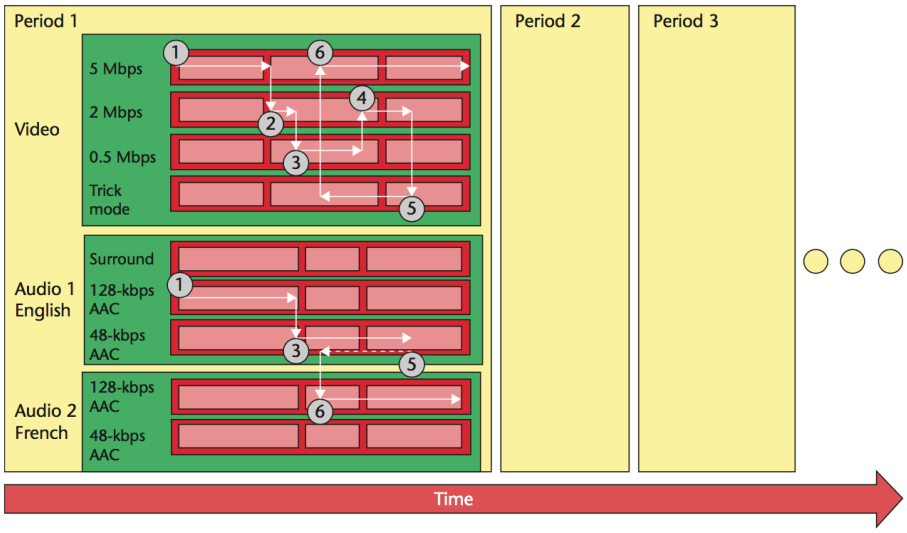


Figura 3.1: Esempio di Dynamic Adaptive Streaming

video che l'audio. Da notare il punto 5: si effettua un passaggio al segmento trick mode ossia la modalità che permette di riprodurre il video all'indietro e ricominciare la riproduzione dal punto 6, effettuando anche uno switch audio in lingua francese. Questo esempio è solo uno dei tanti semplici casi di dynamic streaming, infatti è possibile aggiungere ulteriori feature come 3D multimedia, contenuto con sottotitoli e didascalie, switch da diverse camere view, ecc.

3.2 Caratteristiche dello standard DASH

La figura 3.2 mostra uno schema di funzionamento dello standard MPEG-DASH; la specifica fornisce:

- definizione di Media Presentation definito come collezione strutturata di dati accessibili al Client DASH mediante Media Presentation Description.
- definizione del formato di un segmento, definito come unità di dati fondamentale di un Media Presentation.

- definizione del protocollo usato per l'invio dei segmenti, ossia il protocollo HTTP;

- descrizione informativa su come un client DASH utilizza le informazioni utili per fornire il servizio streaming all'utente.

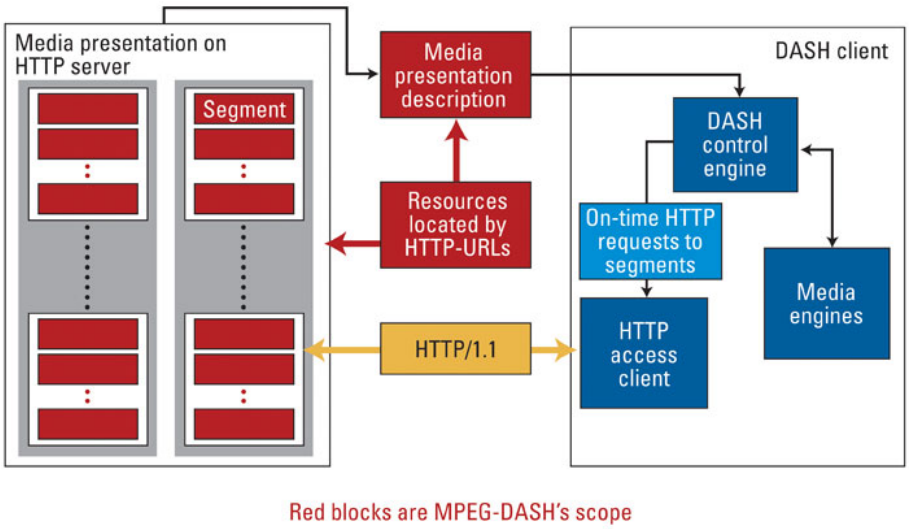


Figura 3.2: Overview MPEG-DASH

3.2.1 Media Presentation

Media Presentation è una collezione strutturata di dati multimediali codificati accessibili al client DASH. Come mostrato in figura 3.3:

- Media Presentation è organizzata in una sequenza di uno o più periodi (Period) consecutivi e non sovrapposti.

- Ogni Period consiste in uno o più unità di contenuto multimediale chiamato segmento (Segment).

- Ogni Segment è disponibile in uno o più formati (Representation).

Una Representation rappresenta l'encoding di un insieme di segmenti e differisce dagli altri insiemi per il bitrate, risoluzione dell'immagine, lingua o codec utilizzati; nella figura 3.3 di esempio, Representation 1 ha una risoluzione di 640x480 pixel e un bitrate di 500 kbit/s, mentre Representation 2 ha la stessa risoluzione, ma un bitrate di 250 kbit/s. Ogni Representation consiste in in un insieme di segmenti. Quest'ultimi rappresentano l'unità di un contenuto multimediale che può essere unicamente referenziata da un HTTP-URL. Inoltre, essi sono indicizzati e insieme formano un flusso multimediale.

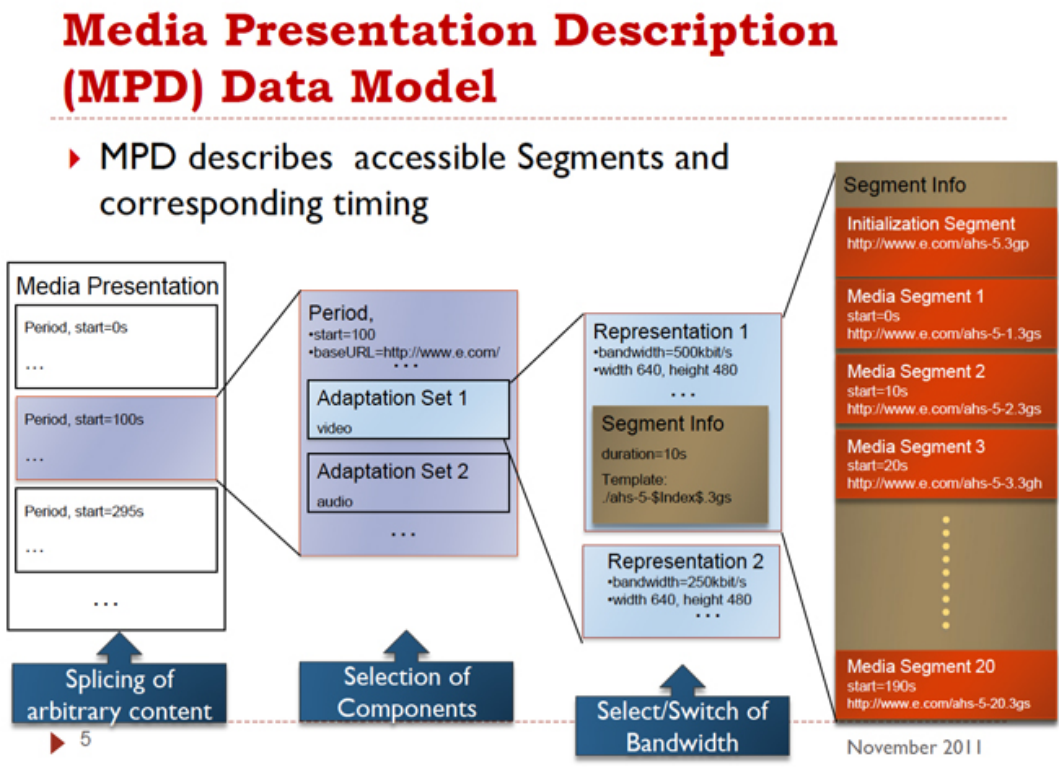


Figura 3.3: Media Presentation Data Model

3.2.2 Media Presentation Description

Il Media Presentation Description (MPD), chiamato anche DASH manifest file, è un file XML (fig 3.4) che descrive le informazioni dei segmenti (time, url, risoluzione video, bitrate, etc...) e può essere organizzato in molti modi diversi, a seconda dei vari casi d'uso [23].

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- MPD file Generated with GPAC version 0.5.1-DEV-rev5379  on 2014-09-10T13:27:38Z-->
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011" minBufferTime="PT1.500000S" type="static" mediaPresentationDuration=
    "PT0H9M56.46S" profiles="urn:mpeg:dash:profile:isoff-on-demand:2011">
  <ProgramInformation moreInformationURL="http://gpac.sourceforge.net">
    <Title>dashed/BigBuckBunny_4s_onDemand_2014_05_09.mpd generated by GPAC</Title>
  </ProgramInformation>
  <Period duration="PT0H9M56.46S">
    <AdaptationSet segmentAlignment="true" group="1" maxWidth="480" maxHeight="360" maxFrameRate="24" par="4:3"
        subsegmentStartsWithSAP="1">
      <Representation id="320x240 45.0kbps" mimeType="video/mp4" codecs="avc1.42c00d" width="320" height="240"
          frameRate="24" sar="1:1" startWithSAP="1" bandwidth="45226">
        <BaseURL>bunny_45226bps/BigBuckBunny_4snonSeg.mp4</BaseURL>
        <SegmentBase indexRangeExact="true" indexRange="885-2716">
          <Initialization range="0-884" />
        </SegmentBase>
      </Representation>
      <Representation id="320x240 89.0kbps" mimeType="video/mp4" codecs="avc1.42c00d" width="320" height="240"
          frameRate="24" sar="1:1" startWithSAP="1" bandwidth="88783">
        <BaseURL>bunny_88783bps/BigBuckBunny_4snonSeg.mp4</BaseURL>
        <SegmentBase indexRangeExact="true" indexRange="887-2718">
          <Initialization range="0-886" />
        </SegmentBase>
      </Representation>
      <Representation id="320x240 129.0kbps" mimeType="video/mp4" codecs="avc1.42c00d" width="320" height="240"
          frameRate="24" sar="1:1" startWithSAP="1" bandwidth="128503">
        <BaseURL>bunny_128503bps/BigBuckBunny_4snonSeg.mp4</BaseURL>
        <SegmentBase indexRangeExact="true" indexRange="887-2718">
          <Initialization range="0-886" />
        </SegmentBase>
      </Representation>
      <Representation id="480x360 177.0kbps" mimeType="video/mp4" codecs="avc1.42c015" width="480" height="360"
          frameRate="24" sar="1:1" startWithSAP="1" bandwidth="177437">
        <BaseURL>bunny_177437bps/BigBuckBunny_4snonSeg.mp4</BaseURL>
        <SegmentBase indexRangeExact="true" indexRange="888-2719">
          <Initialization range="0-887" />
        </SegmentBase>
      </Representation>
      <Representation id="480x360 218.0kbps" mimeType="video/mp4" codecs="avc1.42c015" width="480" height="360"
          frameRate="24" sar="1:1" startWithSAP="1" bandwidth="217761">
        <BaseURL>bunny_217761bps/BigBuckBunny_4snonSeg.mp4</BaseURL>
        <SegmentBase indexRangeExact="true" indexRange="888-2719">
          <Initialization range="0-887" />
        </SegmentBase>
      </Representation>
      <Representation id="480x360 256.0kbps" mimeType="video/mp4" codecs="avc1.42c015" width="480" height="360"
          frameRate="24" sar="1:1" startWithSAP="1" bandwidth="255865">
        <BaseURL>bunny_255865bps/BigBuckBunny_4snonSeg.mp4</BaseURL>
        <SegmentBase indexRangeExact="true" indexRange="888-2719">
          <Initialization range="0-887" />
        </SegmentBase>
      </Representation>
```

Figura 3.4: Media Presentation Description

3.2.3 Client DASH

Il Client DASH rappresenta la componente principale: dopo aver ricevuto il manifest file (MPD), ne effettua un parsing apprendendo le informazioni utili per lo streaming; successivamente si prepara a ricevere il flusso di segmenti audio o/e video dal server scegliendo la representation più appropriata; detto in termini più semplici, il file multimediale è suddiviso in segmenti più piccoli, ogni segmento è disponibile a multipli bitrate ed il compito di un buon client è quello di selezionare il formato più appropriato per ogni segmento.

3.3 Android Overview

L'applicazione è stata sviluppata per Android, famoso e diffuso sistema operativo per dispositivi mobili, sviluppato da Google e basato su kernel Linux [16]. Android è distribuito sotto la licenza Apache che permette di modificare e distribuire liberamente il codice sorgente. Inoltre, Android dispone di una vasta comunità di sviluppatori che realizzano applicazioni con l'obiettivo di aumentare le funzionalità dei dispositivi. Queste applicazioni sono scritte soprattutto in linguaggio di programmazione Java. Google è anche a capo del progetto Open Source Android, il cui obiettivo è quello di creare uno strumento in grado di rendere sempre più piacevole l'utilizzo del dispositivo per gli utenti che lo posseggono [16]. La principale piattaforma hardware per cui Android è stato ingegnerizzato, è l'architettura ARM. L'interfaccia utente è fondata sul concetto di direct manipulation che utilizza degli ingressi singoli o multipli per controllare oggetti visibili sullo schermo. La risposta a questi eventi è stata progettata per essere veloce e per fornire un'interfaccia più fluida. All'interno del dispositivo possiamo inoltre trovare sensori hardware, come accelerometro, giroscopio, sensore di prossimità, sensore di temperatura, ecc., i quali vengono utilizzati da applicazioni che devono rispondere ad azioni svolte dall'utente. Si può trovare un utilizzo comune nei dispositivi Android di questi sensori nella rotazione automatica dello schermo da verticale a orizzontale o viceversa.

3.4 Descrizione dell'applicazione

In questa sezione descriverò parte della progettazione della mia applicazione che implementa un client Dynamic Adaptive Streaming: *DashPlayer*. L'applicazione utilizza le librerie ExoPlayer di Android compatibili con le API 16 di Android, quindi, per le versioni precedenti ad Android 4.1 non è garantita la piena compatibilità. Non è stata data importanza all'interfaccia grafica perché non era rilevante al proseguimento degli scopi di questa tesi.

3.4.1 Librerie Utilizzate

- ExoPlayer library: libreria costruita sopra le API MediaPlayer di Android, fornisce un player customizzabile ed estendibile.
- GraphView: libreria per la costruzione di grafici.

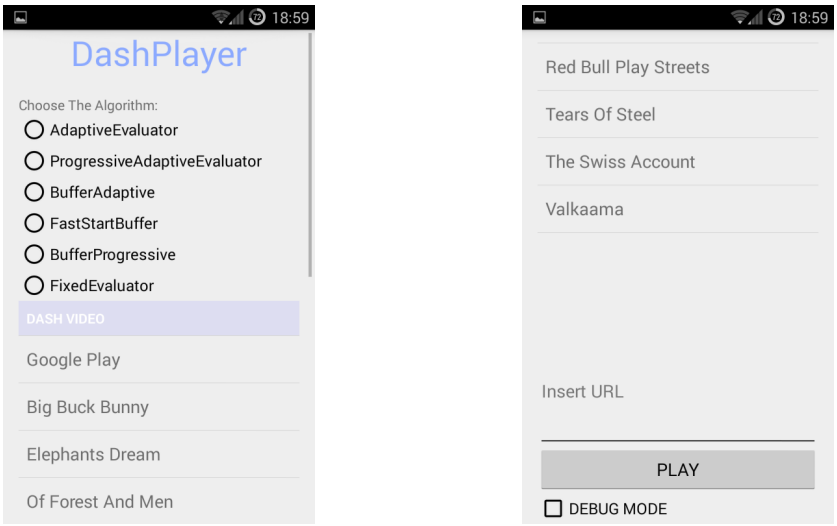
3.4.2 Permessi Richiesti

L'applicazione prima di essere installata nel dispositivo richiede i seguenti permessi:

- Accesso alla rete internet.
- Scrivere sulla memoria esterna.
- Leggere sulla memoria esterna

3.4.3 Funzionamento delle Activity

Un *activity* è quella parte dell'applicazione con cui l'utente può interagire. *DashPlayer* ha un totale di 2 activity: *SelectVideoActivity*, come è possibile osservare dalla figura 3.5a, è l'activity home che permette di selezionare fra i vari video di esempio e di scegliere l'algoritmo per poter svolgere i test. Inoltre scorrendo in basso, è possibile inserire manualmente l'URL del manifest e permettere all'utente di attivare o disattivare la modalità debug; attivando



(a) Parte superiore (b) Parte inferiore

Figura 3.5: Screenshot dell'interfaccia di DashPlayer

questa modalità l'applicazione durante la riproduzione del video salverà i dati su un file di *log* nella memoria del telefono o nella SD se in dotazione fig 3.5b.

Selezionando il tasto play viene invocata la seconda activity della mia applicazione: PlayerActivity di cui possiamo vedere uno screenshot nella figura 3.6. Per l'implementazione di *PlayerActivity* è stato utilizzato parte del codice già presente nella demo di Exoplayer, ma sono state comunque apportante delle modifiche quali:

- Aggiunta di un grafico che si aggiorna in tempo reale sulla qualità del video.

- Aggiunta delle informazioni utili ai fini del debug come dimensione e posizione del buffer in ms.

- Scelta della modalità di log: normale o verbose.

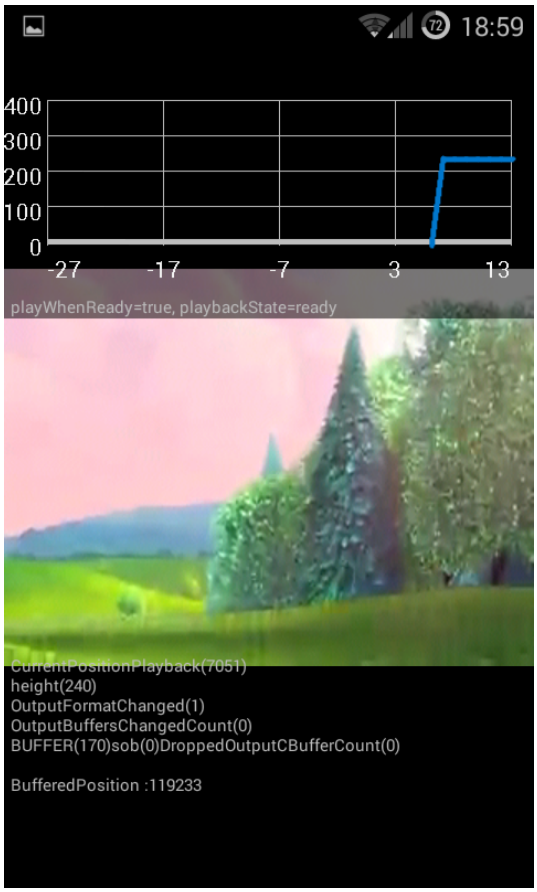


Figura 3.6: Screenshot di PlayerActivity

3.4.4 Logfile

L'applicazione ad ogni sessione di riproduzione crea 2 logfile con nome timestamp_download e timestampt_player contenenti i dati utili per poter procedere alla valutazione sperimentale degli algoritmi; dall'esempio di figura 3.7 ogni riga rappresenta un segmento scaricato e per ogni colonna è possibile avere i seguenti dati (ordinati rispetto alle colonne del logfile):

1. tipo di connessione (3g/Wi-Fi);
2. nome del video riprodotto;
3. duration del segmento (s);
4. inizio del segmento (ms);

```
Wi-Fi  BigBuckBunny 2s 0 2000 5.62 358 13000 290502 45652 230 1770 1 1 20150621203724
Wi-Fi  BigBuckBunny 2s 2000 4000 7.75 227 18428 649444 89283 1906 2094 2 1 20150621203726
Wi-Fi  BigBuckBunny 2s 4000 6000 8.72 729 34675 380521 131087 1906 4094 3 1 20150621203727
Wi-Fi  BigBuckBunny 2s 6000 8000 10.00 1038 49311 380046 178351 2514 5486 4 1 20150621203728
Wi-Fi  BigBuckBunny 2s 8000 10000 11.89 1733 61481 283813 221600 4420 5580 5 1 20150621203730
Wi-Fi  BigBuckBunny 2s 10000 12000 13.31 1299 62189 382996 221600 5837 6163 6 1 20150621203732
Wi-Fi  BigBuckBunny 2s 12000 14000 14.66 1128 64774 459390 262537 7180 6820 7 1 20150621203733
Wi-Fi  BigBuckBunny 2s 14000 16000 17.29 2437 98115 322084 334349 9810 6190 8 1 20150621203736
Wi-Fi  BigBuckBunny 2s 16000 18000 18.23 828 61298 592251 262537 10754 7246 9 1 20150621203737
Wi-Fi  BigBuckBunny 2s 18000 20000 19.77 1447 58617 324074 334349 12295 7705 10 1 20150621203738
Wi-Fi  BigBuckBunny 2s 20000 22000 20.90 1025 59300 462829 262537 13424 8576 11 1 20150621203739
Wi-Fi  BigBuckBunny 2s 22000 24000 21.84 821 57816 563371 334349 14351 9649 12 1 20150621203740
Wi-Fi  BigBuckBunny 2s 24000 26000 24.92 2899 120967 333817 396126 17433 8567 13 1 20150621203743
Wi-Fi  BigBuckBunny 2s 26000 28000 26.37 1361 95986 564208 334349 18889 9111 14 1 20150621203745
Wi-Fi  BigBuckBunny 2s 28000 30000 31.05 4572 120231 210377 396126 23557 6443 15 1 20150621203749
Wi-Fi  BigBuckBunny 2s 30000 32000 34.53 2327 98103 337268 334349 27063 4937 16 1 20150621203753
Wi-Fi  BigBuckBunny 2s 32000 34000 37.93 3175 68969 173780 262537 30460 3540 17 1 20150621203756
Wi-Fi  BigBuckBunny 2s 34000 36000 38.80 750 55007 586741 221600 31304 4696 18 1 20150621203757
Wi-Fi  BigBuckBunny 2s 36000 38000 39.38 482 72594 1204879 262537 31895 6105 19 1 20150621203758
Wi-Fi  BigBuckBunny 2s 38000 40000 40.14 685 98884 1154849 334349 32667 7333 20 1 20150621203759
Wi-Fi  BigBuckBunny 2s 40000 42000 40.72 509 121970 1917013 396126 33252 8748 21 1 20150621203759
Wi-Fi  BigBuckBunny 2s 42000 44000 43.12 2247 144112 513082 522286 35657 8343 22 1 20150621203801
Wi-Fi  BigBuckBunny 2s 44000 46000 45.09 1156 117268 811543 396126 37626 8374 23 1 20150621203803
Wi-Fi  BigBuckBunny 2s 46000 48000 46.25 1053 147567 1121116 522286 38764 9236 24 1 20150621203805
Wi-Fi  BigBuckBunny 2s 48000 50000 47.26 890 156593 1407577 595491 39791 10209 25 1 20150621203806
Wi-Fi  BigBuckBunny 2s 50000 52000 48.15 710 207436 2337307 791182 40678 11322 26 1 20150621203807
Wi-Fi  BigBuckBunny 2s 52000 54000 49.46 1159 284115 1961104 1032682 41986 12014 27 1 20150621203808
Wi-Fi  BigBuckBunny 2s 54000 56000 51.63 1810 381235 1685016 1244778 44159 11841 28 1 20150621203810
Wi-Fi  BigBuckBunny 2s 56000 58000 52.53 792 193633 1955888 1244778 45049 12951 29 1 20150621203811
Wi-Fi  BigBuckBunny 2s 58000 60000 53.31 658 183387 2229629 1546902 45839 14161 30 1 20150621203812
Wi-Fi  BigBuckBunny 2s 60000 62000 54.55 1165 265513 1823265 1546902 47077 14923 31 1 20150621203813
Wi-Fi  BigBuckBunny 2s 62000 64000 58.08 3406 291784 685341 1546902 50594 13406 32 1 20150621203816
Wi-Fi  BigBuckBunny 2s 64000 66000 59.96 1781 326177 1465140 1244778 52474 13526 33 1 20150621203818
Wi-Fi  BigBuckBunny 2s 65999 67999 63.22 3164 363189 918303 1244778 55745 12254 34 1 20150621203822
Wi-Fi  BigBuckBunny 2s 68000 70000 66.58 3267 286797 702288 1032682 59108 10892 35 1 20150621203825
Wi-Fi  BigBuckBunny 2s 70000 72000 68.10 1435 170517 950617 791182 60627 11373 36 1 20150621203826
Wi-Fi  BigBuckBunny 2s 72000 74000 70.09 1855 188893 814632 791182 62597 11403 37 1 20150621203828
Wi-Fi  BigBuckBunny 2s 74000 76000 72.09 1897 159345 671987 595491 64599 11401 38 1 20150621203830
```

Figura 3.7: Esempio di log

5. fine del segmento (ms);
6. tempo di sessione (s);
7. tempo di download (ms);
8. segment size (bytes);
9. throughput (bps);
10. bitrate(bps);
11. playback position (ms);
12. livello del buffer (ms);
13. segment Index;
14. Algorithm Id;
15. Timestamp;

Invece ts_player è un logfile al livello del player: ogni riga rappresenta un secondo di video. I due logfile saranno molti importanti più avanti nel mio lavoro di tesi perché costituiranno i dati "grezzi" da cui partire per compiere le mie valutazioni. Per ulteriori informazioni sul significato dei dati consultare la documentazione di DashPlayer inclusa nel sorgente del software.

Capitolo 4

Analisi sperimentale

Nel sistema Dynamic Adaptive Streaming over Http il server contiene diverse representation dello stesso video, ossia formati diversi codificati in differenti bitrate e livelli di qualità. Il client può richiedere i segmenti del video in vari formati disponibili. Il compito di un algoritmo di rate adaptation è quello di fornire al client la "scelta" da effettuare prendendo in input alcuni dati e ritornando in output la representation da cui scaricare il segmento successivo. Appare evidente come il processo algoritmico di decisione della representation ottimale per ogni segmento sia un elemento chiave ed una delle più grandi sfide nel sistema dynamic adaptive streaming. In questo capitolo, porterò a termine alcune valutazioni sperimentali su 3 algoritmi, tramite l'ausilio dell'app DashPlayer.

4.1 Metriche e criteri di valutazione

Basandomi sulle preferenze e i bisogni degli utenti analizzati nell'articolo [17], posso affermare che un algoritmo di rate adaptation compie bene il suo lavoro se riesce a raggiungere i seguenti *goal*:

i. ridurre al minimo le interruzioni del playback durante la riproduzione;

ii. massimizzare la qualità media dello stream;

iii. minimizzare il numero di *shift* della qualità;

iv. minimizzare il *delay* tra la richiesta dell'utente e l'inizio effettivo del playback;

Da notare che i goal (ii) e (iii) costituiscono un *trade-off* tra scegliere sempre la qualità migliore possibile, qualora la connessione sia poco stabile, e decidere di rimanere con una qualità costante evitando le continue fluttuazioni del bitrate, fastidiosi agli occhi dell'utente. Quindi, saranno utilizzate le seguenti metriche per compiere le valutazioni:

- average bitrate del video, ovvero una stima della qualità media offerta all'utente;
- livello del buffer;
- tempo in secondi delle pause (interruzioni del playback);
- numero di switch del bitrate effettuati;

4.1.1 Notazioni utilizzate

Nelle pagine seguenti, utilizzerò il termine representation per riferirmi all'insieme dei segmenti continui del video, codificati allo stesso bitrate [17]. Inoltre, utilizzerò: *Rmax* (*Rmin*) per indicare la representation con il massimo (minimo) average bitrate fra tutte quelle disponibili; R↑ (R↓) per riferirmi alla representation con il successivo più alto (più basso) average bitrate rispetto a R. Denoterò con τ il throughput istantaneo ed assumerò che: (1) il client esegua il download dei segmenti in ordine cronologico, (2) ogni segmento sia scaricato solamente in una representation, (3) ogni segmento i-1 debba essere scaricato completamente prima dell'inizio del download del segmento i. Definisco il throughput istantaneo (τ) come l'effettiva quantità di byte trasmessi in una determinata quantità di tempo:

$$\frac{\textit{Dim. ultimo Pacchetto Scaricato}}{\textit{tempo di download}}$$

Invece, definisco β come il buffer del video in secondi rispetto allo stato corrente del playback.

4.2 Descrizione degli algoritmi

Gli algoritmi seguenti sono ispirati da altri già esistenti in letteratura [17][18]. In appendice possiamo trovare gli algoritmi scritti in pseudo-codice, mentre, per l'implementazione rimando alla repository github dell'app DashPlayer [22]. Gli algoritmi valutati, da un lato, hanno diversi approcci su come calcolare la representation del segmento successivo, dall'altro fanno affidamento su, all'incirca, gli stessi dati. Per raggiungere il goal (iv), all'avvio del playback quando ancora non si conosce il throughput istantaneo τ, tutti gli algoritmi restituiscono in output *Rmin*. Ho dato agli algoritmi i seguenti nomi: *Stepwise Buffer Unaware* (SBU), *Sudden Buffer Aware* (SBA) e *Thresold Buffer Aware* (SBU).

4.2.1 Stepwise Buffer Unaware

È l'algoritmo *dummy*: aumenta la qualità progressivamente basandosi solo sul throughput τ. Se τ rimane costante rispetto al bitrate corrente, allora, l'algoritmo ritorna la representation corrente, se il throughput è aumentato l'algoritmo ritorna la representation successiva con il bitrate più alto, o più bassa in caso contrario.

4.2.2 Sudden Buffer Aware

Questo algoritmo prende in input τ, β e due valori di *threshold* del buffer espressi in secondi; dopo essersi calcolato τ, sceglie la representation più alta possibile (*suddenly*), in modo non stepwise, ma solo dopo aver controllato che il livello di β sia maggiore di *BufIncrease*. Allo stesso modo, prima di decrementare, controlla che il livello di β sia maggiore di *BufferDecrease*, se lo è, aspetta che scenda sotto quel livello prima di decrementare. L'algoritmo

era presente nella demo di Exoplayer ed i valori di *BufferDecrease* e *BufferIncrease* erano stati configurati, rispettivamente, con 10000 e 18000; quindi, ho utilizzato questi valori per la valutazione.

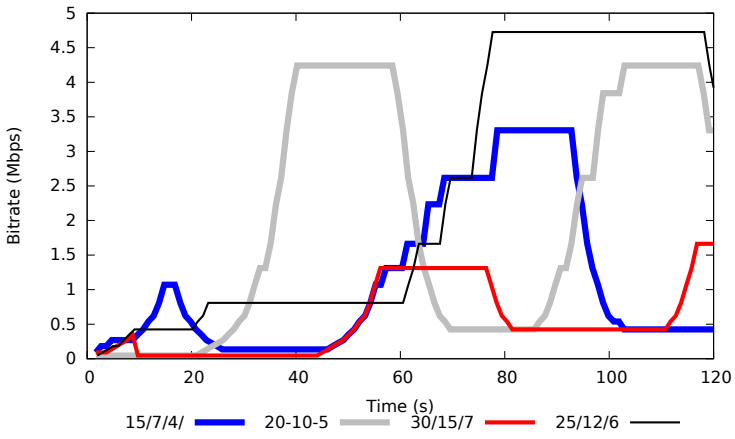
4.2.3 Threshold Buffer Aware

È la mia proposta di algoritmo che mira a tenere stabile il bitrate cercando di mantenere alto il livello del buffer. TBA utilizza tre soglie (Low, Mid e High) per monitorare il buffer ed evitare interruzioni nel playback. Se β < Low, l'algoritmo sceglie la representation con il bitrate minimo, in questo modo si cercano di evitare le pause del playback quando il buffer scende sotto la soglia Low. Se β < Mid, sceglie la representation R↓. Se il buffer si trova tra Mid e High mantiene la representation corrente. Infine, se β > High, sceglie R↑, dopo aver verificato che il valore di τ permetta di aumentare il bitrate. Inoltre, nella fase iniziale, quando ancora non si è certi dell'andamento reale del throughput, è stata inserita una "fase iniziale accelerata" in cui l'algoritmo cerca di aumentare la qualità più velocemente. In seguito, esce dalla fase accelerata tornando al funzionamento classico descritto sopra, appena raggiunge la qualità massima o τ non permette di continuare ad incrementare il bitrate.

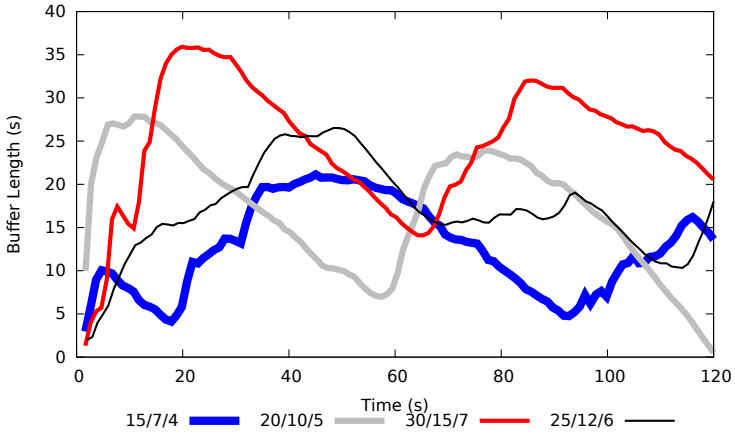
Setting delle variabili di threshold

Ho effettuato delle analisi preliminari sull'algoritmo TBA, provando varie combinazioni con le variabili threshold. Ho testato SBA in un scenario con segnale Wi-Fi instabile. L'obiettivo era decidere l' "assetto" migliore, ovvero, trovare un compromesso tra cercare di ottenere la migliore qualità possibile, evitando troppe fluttuazioni del bitrate e mantenere un livello di buffer sufficiente a non far interrompere il playback. Dopo aver eseguito i test in uno scenario con scarso segnale Wi-Fi, utilizzando varie configurazioni, ho voluto utilizzare la configurazione più prudente, ovvero quella che: (1) tenesse alto il livello di β, (2) mantenesse una qualità costante (3) evitasse le pause del playback. Da fig. fig 4.1 si può evincere che sia per il buffer sia per stabilità

del bitrate l'assetto 30, 15 e 7 è quella che rispetta maggiormente i punti (1), (2) e (3).



(a) Bit-rate detected



(b) Buffer detected

Figura 4.1: Test dell'algoritmo SBA con diverse variabili di threshold

4.3 Test degli algoritmi

Per effettuare i test ho utilizzato il video BigBuckBunny disponibile sul server di *Itec* all'indirizzo [20]. Le representation sono in totale 20, quindi, con altrettanti livelli di bitrate disponibili: R=45652, 89283,...4219897, 4726737 (kbit/s); la duration di ogni segmento è 1 secondo. I test sono stati

effettuati in diversi scenari e ripetuti diverse volte al fine di poter compiere una valutazione sugli algoritmi nel modo più affidabile possibile.

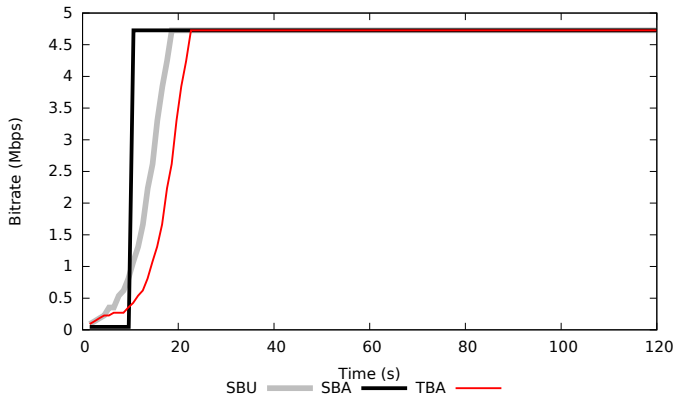
4.3.1 Wifi Max

Name	Average Bitrate	Average Switches	Pause Time
Unit	[kbps]	[Number of Switches]	[Second]
SBU	4249	5	0
SBA	4372	2	0
TBA	4100	5	0

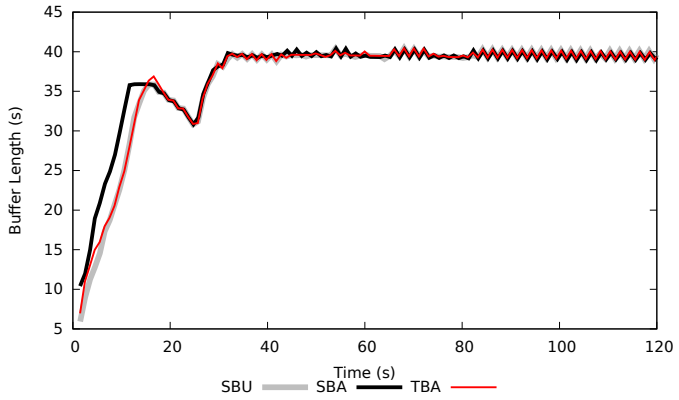
Tabella 4.1: Comparison scenario max Wi-Fi

In questo scenario, i test sono stati effettuati in un punto della mia abitazione molto vicino all'access point. Non si tratta di uno scenario particolarmente interessante, ma ho deciso di valutarlo per completezza e anche per dimostrare la correttezza sia degli algoritmi che dell'app DashPlayer. Il video è stato riprodotto per 1 minuto ciascuno, una volta, per ogni algoritmo. Possiamo osservare, in fig 4.2a, come tutti gli algoritmi raggiungano la massima qualità, con un bit-rate di 4726737 kbit/s, entro i venti secondi. SBA è stato l'algoritmo più veloce a salire, poiché, a differenza degli altri aumenta la qualità di colpo, non in modo progressivo. Il più lento a salire è stato TBA che si dimostra l'algoritmo più "prudente", poiché, aspetta che il Throughput si sia stabilizzato, prima di poter aumentare il bit-rate del segmento successivo. La figura 4.2b mostra l'andamento del buffer: possiamo osservare un calo intorno ai 10 e 30 secondi, dovuto ad un effettivo ridimensionamento del throughput che può essere dovuto alla "pesantezza" dei primi segmenti del video [20]. Successivamente, il buffer mostra la sua proprietà di convergenza, stabilizzandosi sui 40 secondi. Dalla tabella 4.1, si può ancora osservare come SBA aumenti rapidamente la qualità in soli 2

switch, mentre, SBA e TBA mantengono più o meno un'andamento simile in scenari con un'ottimo segnale Wi-FI.



(a) Bit-rate detected



(b) Buffer detected

Figura 4.2: Bit-rate e buffer rilevati in uno scenario con un ottimo segnale Wi-Fi

4.3.2 Critical Wifi

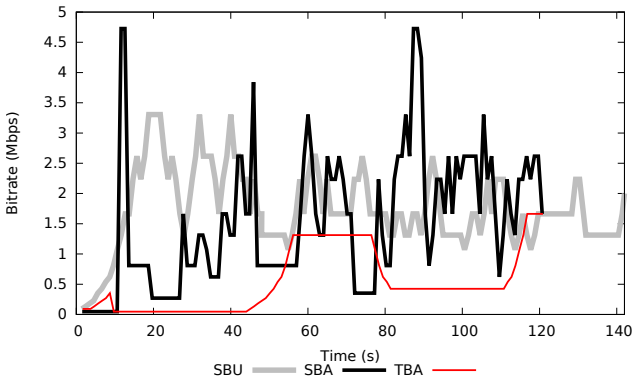
Ho effettuato questi test in un punto della mia abitazione con un segnale Wi-fi molto scarso, al fine di poter valutare il comportamento degli algoritmi in condizioni instabili della rete Wi-Fi. Questa volta ho riprodotto il video per 2 minuti tenendo lo smartphone fermo nel punto critico individuato. Da notare come sull'asse delle ascisse, il dato *Time* si riferisca al tempo

di riproduzione totale, indipendente dallo stato del playback. Studiando i risultati ottenuti in fig. 4.3, si può osservare che la linea del bitrate di SBU supera i 120 secondi riproduzione. Questo tempo aggiuntivo indica che l'algoritmo ha impiegato più tempo a terminare il playback perché ha subito qualche interruzione durante la sessione di streaming.

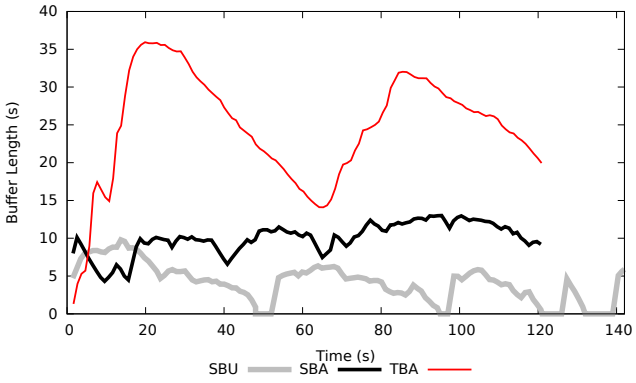
Name	Average Bitrate	Average Switches	Pause Time
Unit	[kbps]	[Number of Switches]	[Second]
SBU	1752	29	$23,5$
SBA	1531	33	0
TBA	536	10	0

Tabella 4.2: Comparison scenario critical Wi-Fi

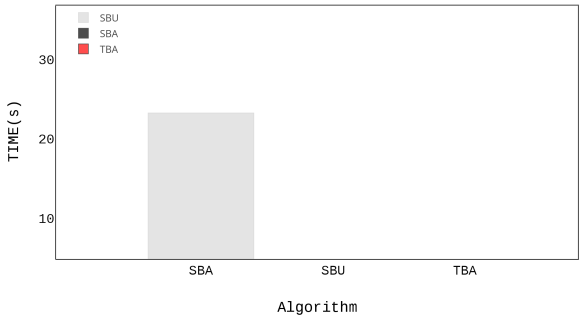
Dal punto di vista della qualità offerta, SBU e SBA possono offrire una qualità migliore a discapito della stabilità del video provocando cambi di risoluzione repentini 4.3a. Al contrario, TBA, pur non offrendo mai un bit-rate superiore ai 2 Mbps, offre una maggiore stabilità e un minor numero di switch del bitrate (oss. tab. 4.2) effettuando meno switch di risoluzione (quasi tre volte in meno) rispetto a SBU e SBA. Osservando fig. 4.3b, possiamo notare le pause di SBU nei punti in cui la dimensione buffer è scesa a zero, mentre, nella tabella 4.2 abbiamo il tempo totale delle pause di SBU. Il buffer di TBA non scende mai sotto la soglia dei 15 secondi, poiché ho configurato le variabili di threshold sui valori 30-15-7. Il buffer di SBA tende a stare fra i 10 e 15 secondi, mentre SBU non monitorando il buffer soffre di continui stalli e re-buffering.



(a) Bit-rate detected



(b) Buffer detected



(c) Pausa totale del playback

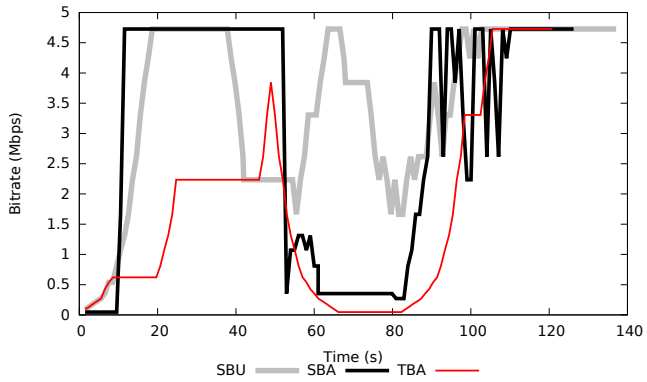
Figura 4.3: Bit-rate, buffer e pause rilevati in uno scenario con uno scarso segnale Wi-Fi

4.3.3 Mobility Wifi

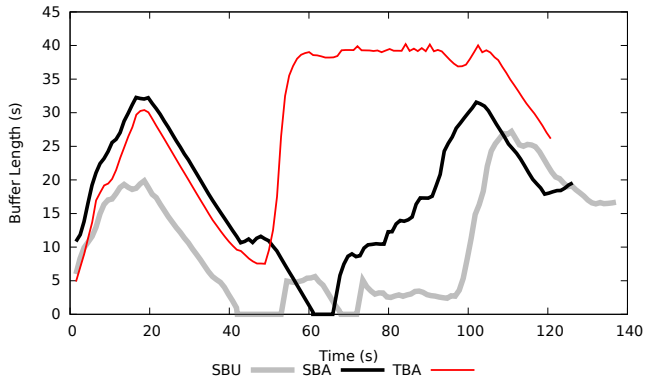
Il test di mobilità consiste nell'effettuare degli spostamenti, da un punto A vicino all' access point in cui il segnale è molto forte, al punto critico C. In questo scenario si vuole testare la capacità degli algoritmi di adattarsi alle improvvise fluttuazioni del throughput. Ho eseguito il test in questo modo: 10 secondi nel punto A, 10 secondi di transizione dal punto A al punt C, 30 secondi nel punto C e ritorno al punto A. Ho ripetuto questa procedura due volte ottenendo un totale di 120 secondi di playback. Oss. il grafico in fig 4.4a: apparentemente, l'algoritmo SBU e SBA sembrano offrire un ottimo bit-rate seppur repentino, mentre TBA sembrerebbe non essere alla pari degli altri due; andando ad osservare fig 4.4b, invece possiamo constatare che il livello del buffer non è mai sceso a zero per l'algoritmo TBA, mentre lo è stato tragicamente per SBU (tra i 40 e 50 secondi) ed anche per SBA (tra i 60 e 70 secondi). Come nello scenario precedente, SBU e SBA sembrano mantenere un average bitrate superiore di TBA (tab 4.3), ma tutto questo a discapito del buffer. Per quanto riguarda le pause (oss. tab. 4.3), SBU e SBA sono stati gli algoritmi che hanno subito interruzioni del playback; SBU ha subito la pausa più consistente, mentre durante la sessione di TBA non sono state rilevate pause.

Name	Average Bitrate	Average Switches	Pause Time
Unit	[kbps]	[Number of Switches]	[Second]
SBU	3549	11	$17,58$
SBA	3059	8	$6,60$
TBA	1701	13	0

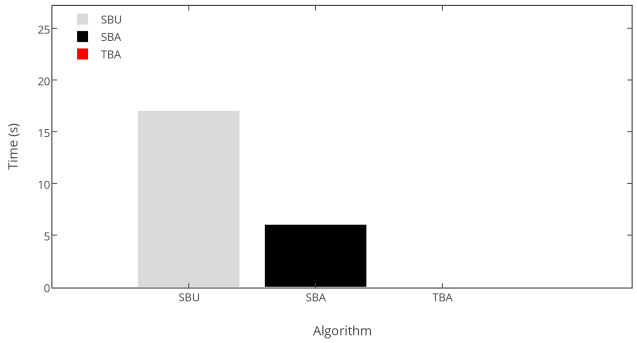
Tabella 4.3: Comparison scenario mobility Wi-Fi



(a) Bit-rate detected



(b) Buffer detected



(c) Pausa totale del playback

Figura 4.4: Bitrate, buffer e pause rilevati in uno scenario di mobilità Wi-Fi

4.4 Risultati delle valutazioni

l'algoritmo TBA sembra essere l'algoritmo migliore, poiché ha assunto il comportamento migliore in tutti gli scenari. L'algoritmo peggiore nei test è risultato essere SBU, come si sarebbe potuto aspettare visto che riceve un informazione in meno rispetto a SBA e TBA, ovvero non conosce la dimensione del buffer. Questo ci fa capire come sia il livello del buffer sia il throughput siano dei dati importanti per poter formulare algoritmi efficienti. L'algoritmo SBA non ha dimostrato grossi problemi nello scenario con Wi-Fi instabile mantenendo una qualità media superiore rispetto a TBA, vincendo sul goal (ii), ma perdendo per (iii) visto il gran numero di switch effettuati. Posso concludere che TBA riesce a raggiungere il maggior numero di goal.

Conclusioni e sviluppi futuri

In questa tesi è stato illustrato lo standard ISO MPEG-DASH per le tecnologie dynamic adaptive streaming ed è stata presentata la mia proposta applicativa: DashPlayer. L'applicazione implementa un client conforme allo standard DASH su Android e permette di raccogliere dati sperimentali sulla qualità del servizio di streaming offerto. Nel quarto capitolo, è stato illustrato come il buon funzionamento di un client DASH dipenda dall'efficienza di particolari algoritmi, detti di rate adaptation.

Ho concluso il mio lavoro valutando sperimentalmente tre di questi algoritmi in diversi scenari. Dai risultati ottenuti un algoritmo si è dimostrato essere più efficiente rispetto agli altri.

Sui possibili sviluppi futuri:

- trovare algoritmi efficienti effettuando ulteriori test su DashPlayer;
- integrare l'applicazione con il sistema Microcast che usa le risorse di tutti gli smartphone connessi alla stessa rete per migliorare l'esperienza di streaming [21];
- porting dell'applicazione su altri sistemi operativi mobili come iOS e WP;
- testare gli algoritmi in diversi scenari: handover, 3G, viaggi in treno, auto, ecc...;

Appendice A

Pseudo-codice degli algoritmi

Input: τ
Output: Representation to be selected for the download of the next segment
current $\leftarrow$ current bitrate representation;
ideal $\leftarrow$ The best bitrate representation having throughput τ;
isHigher $\leftarrow$ current $>$ ideal;
isLower $\leftarrow$ current $<$ ideal;
if *isHigher* **then**
  current $\leftarrow$ ideal ;
else
  let current unchanged;
end
if *isLower* **then**
  current $\leftarrow$ ideal ;
else
  let current unchanged;
end

Algorithm 1: Pseudo-codice dell' Algoritmo Stepwise Buffer Unaware

Input: τ, β, *BufIncrease*, *BufferDecrease*
Output: Representation to be selected for the download of the next segment
current $\leftarrow$ current bitrate representation;
ideal $\leftarrow$ The best bitrate representation having throughput τ;
isHigher $\leftarrow$ current $>$ ideal;
isLower $\leftarrow$ current $<$ ideal;
if *isHigher* **then**
  if $\beta >$ *BufIncrease* **then**
    current $\leftarrow$ ideal ;
  else
    let current unchanged;
  end
end
if *isLower* **then**
  if $\beta <$ *BufDecrease* **then**
    current $\leftarrow$ ideal ;
  else
    let current unchanged;
  end
end

Algorithm 2: Pseudo-codice dell' Algoritmo Sudden Buffer Aware

Input: τ, β, LOW, MID, HIGH, α

Output: Representation to be selected for the download of the next segment

current $\leftarrow$ current bitrate representation;

if $\beta < LOW$ **then**
  return *Rmin*
end
if $\beta < MID$ **then**
  if *current* $> \beta$ **then**
    return R$\downarrow$;
  else
    let current unchanged;
  end
end
if $\beta < HIGH$ **then**
  let current unchanged;
end
if $\beta >$ *HIGH and* **then**
  if *current* == *Rmax and* $R\uparrow >= \tau$ **then**
    let current unchanged;
  else
    return R$\uparrow$;
  end
end

Algorithm 3: Pseudo-codice dell' Algoritmo Threshold Buffer Aware

Appendice B

Dettagli Implementativi

In questa capitolo della testi approfondirò alcuni dettagli implementantivi. Il codice utilizzato è Java; come ambiente di sviluppo è stato utilizzato Android Studio.

B.1 ExoPlayer

ExoPlayer library [19] è una libreria open-source sviluppata da google per dare la possibilità di implementare feature non supportate dalle classiche API del MediaPlayer di Android, come DASH e SmoothStreaming. Inoltre un client che utilizza le librerie ExoPlayer risulta facilmente customizzabile ed estendibile. Ho deciso di utilizzare Exoplayer come libreria per poter scrivere la mia applicazione per i seguenti motivi:

- Supporto per Dynamic Adaptive Streaming Over HTTP (DASH) e SmoothStreaming, nessuno dei due supportati dal framework MediaPlayer.
- Possibilità di essere esteso, customizzabile in funzione dei propri casi d'uso.
- compatibile con gran parte dei dispositivi android.

Gli unici svantaggi che ho potuto rilevare sono:

- I componenti standard audio e video dipendono dalle API MediaCodec, che è stata rilasciata in Android 4.1 (API livello 16). Quindi l'applicazione non funzionerà correttamente su una versione precedente di Android.

- Possibilità di essere esteso, customizzabile in funzione dei propri casi d'uso.

- Exoplayer non riesce a riconoscere automaticamente il formato del media che è deve essere riprodotto. Quindi, l'applicazione deve essere in grado di conoscere il formato per poter costruire una classe ExoPlayer in grado di poterlo riconoscere.

B.2 Library overview

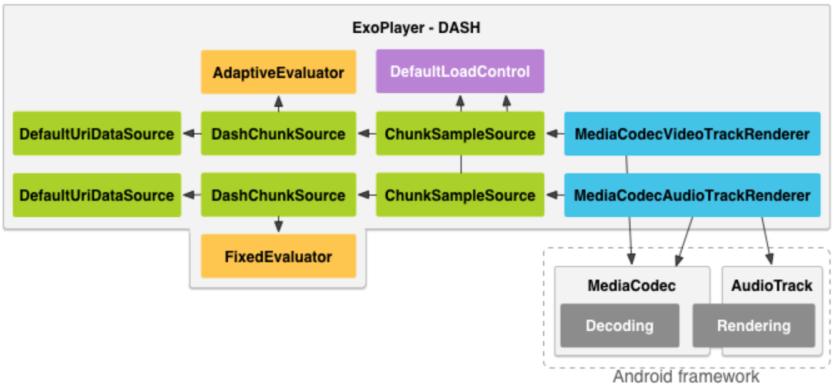


Figura B.1

Nel cuore della libreria Exoplayer si trova la classe *ExoPlayer*. Questa classe mantiene il controllo globale sullo stato del player ma non conosce nulla sua natura del video che deve essere riprodotto o su come deve essere bufferizzato. Inoltre, la libreria è stata progettata facendo una largo uso del pattern Dependency Injection per migliorarne la testabilità e lo sviluppo.

ExoPlayer si appoggia alle API del framework Android di basso livello come *MediaCodec* e *AudioTrack*. Entrambe queste classe richiedono un oggetto *SampleSource*, dalla quale ottengono dei media sample o "campione di dati" pronti per essere riprodotti. Per ottenere un *SampleSource* le istanze delle classi *DataSource* e *extractor* sono "iniettate" nella classe *ExtractorSampleSource* per permettere rispettivamente di scaricare la risorsa multimediale e di estrarre il media sample dai dati scaricati. In questo caso *DefaulturiDataSource* e *Mp4Extractor* sono utilizzati per riprodurre per estrarre dei file MP4 scaricati dal loro URI. Riassumendo, possiamo riassumere che ogni istanza di Exoplayer è costruita iniettando componenti che forniscono le funzionalità richieste dagli sviluppatori. Questo pattern rende più facile la progettazione di players per gli specifici casi d'uso e per la customizzazione dei componenti. Nelle prossime sezioni descriverò tre delle più importanti interfacce in questo modello: *TrackRenderer*, *SampleSource* e *DataSource*.

B.2.1 TrackRenderer

Una classe *Renderer* è in grado di riprodurre uno specifico tipo di media, sia video, audio che di testo. La libreria Exoplayer fornisce *MediaCodecVideoTrackRenderer* come implementazione di default per il rendering video e MediaCodecAudioTrackRenderer per l'audio. Entrambe le implementazioni usano la classe Android MediaCodec per decodificare i media sample. In questo modo possono gestire tutti i formati audio e video supportati da quasi tutti i device Android. Inoltre la libreria fornisce un'implementazione per il rendering del testo, TextTrackRenderer. Il codice qui sotto è un esempio di come instanziare una classe Exoplayer per riprodurre video e audio usando le implementazioni di default:

```
// 1. INSTANZIA IL PLAYER.
player = ExoPlayer.Factory.newInstance(RENDERER_COUNT);
// 2. COSTRUISCE I RENDERERS.
MediaCodecVideoTrackRenderer videoRenderer = ...
MediaCodecAudioTrackRenderer audioRenderer = ...
```

```
// 3. INIETTA I RENDERES ATTRAVERSO IL COSTRUTTORE.
player.prepare(videoRenderer, audioRenderer);
player.sendMessage(videoRenderer,
    MediaCodecVideoTrackRenderer.
    MSG_SET_SURFACE, surface);
// 5. INIZIA LA RIPRODUZIONE.
player.setPlayWhenReady(true);
...
player.release(); // Don't forget to release when done!
```

B.2.2 SampleSource

La classe SampleSource fornisce informazioni sul formato e rende i media sample pronti per essere renderizzati. L'implementazione standard di *TrackRenderer* richiede che delle istanze di *SampleSource* siano iniettate nei propri costruttori. ExoPlayer fornisce delle implementazioni concrete per i differenti utilizzi:

- *ExtractorSampleSource* utilizzata per formati come:
  - MP3
  - M4A
  - MP4
  - WebM
  - MPEG-TS
  - AAC
- *ChunkSampleSource* per:
  - MP3
  - M4A
- *HlsSampleSource* per:

– HLS playback

B.2.3 DataSource

L'implementazione standard di *SampleSource* utilizza delle istanze di *DataSource* per caricare i dati media. Le più comuni implementazioni sono:

- *DefaultUriDataSource* - Per riprodurre i media che scaricati dalla rete.
- *AssetDataSource* - Per riprodurre i media depositati nella cartella *assets* dell'applicazione.

B.2.4 Adaptive MediaPlayback

ExoPlayer supporta l' adaptive streaming attraverso l'uso dell' interfaccia ChunkSampleSource che ha il compito di estrarre dai "chunk" scaricati un campione video (sample) creando un unico flusso multimediale. ChunkSampleSource ha bisogno di un ChunkSource che viene creato dalla classe DashChunkSource. DashChunkSource richiede l' implementazione di un FormatEvaluator e DataSource. FormatEvaluator ha il compito di scegliere fra i vari formati disponibili e quindi è l'interfaccia che permette di implementare algoritmi di rate adaptation. DataSource si occupa invece di mandare richieste GET Http per scaricare i dati. Infine ChunkSampleSource richiede un oggetto LoadControl che si occupa di controllare le politiche di buffering.

Il codice seguente è una parte dell'implementazione che si trova nella classe *DashRendererBuilder*: *manifestFetcher* è l'oggetto che ha la responsabilità di effettuare il parsing del manifest file mentre formatEvaluator è l'oggetto che implementa gli algoritmi di rate adaptation poiché ha il compito di selezionare fra tutti i formati disponibili prima di richiedere il download di un chunk video.

```
LoadControl loadControl = new DefaultLoadControl(new
    DefaultAllocator(BUFFER_SEGMENT_SIZE));
```

```
DefaultBandwidthMeter bandwidthMeter = new DefaultBandwidthMeter();

// Build the video renderer.
videoDataSource = new DefaultUriDataSource(...);
videoChunkSource = new DashChunkSource(manifestFetcher,
    videoAdaptationSetIndex,
    videoRepresentationIndices, videoDataSource, new
        AdaptiveEvaluator(bandwidthMeter),
    LIVE_EDGE_LATENCY_MS, elapsedRealtimeOffset, null, null);
videoSampleSource = new ChunkSampleSource(videoChunkSource,
    loadControl,
    VIDEO_BUFFER_SEGMENTS * BUFFER_SEGMENT_SIZE, true);
videoRenderer = new MediaCodecVideoTrackRenderer(videoSampleSource,
    MediaCodec.VIDEO_SCALING_MODE_SCALE_TO_FIT);
```

Bibliografia

[1] Thomas Barnett, 2015, *Cisco VNI Complete Forecast Update: Key Trends Include Mobility, M2M and Multimedia Content*

[2] Jan Ozer, 2011, *What is Streaming? A high-level view of streaming media technology, history, and the online video market landscape*

[3] Farrel, Adrian, Morgan Kaufmann, 2004, *The Internet and its protocols*

[4] Microsoft Support, 2014, *The OSI Model's Seven Layers Defined and Functions Explained*

[5] Dave Nelson, 2008, *Windows Media Server or Web Server*

[6] Aliprandi, 2010, *Apriti standard! Interoperabilità e formati aperti per l'innovazione tecnologica*

[7] Douglas E. Comer, 2006, *Internetworking with TCP/IP - Principles, Protocols and Architecture*

[8] Ozer, 2010, *Look: H.264 and VP8 Compared*

[9] K. Sayood, 2006, *Introduction to Data Compression*

[10] Gannes, Liz, 10 June 2009, *The Next Big Thing in Video: Adaptive Bitrate Streaming*

[11] Christopher Muller, Stefan Lederer, Cristian Timmerer, 2012, *An Evaluation of Dynamic Adaptive Streaming over HTTP in Vehicular Environments*

[12] Thomas Stockhammer, 2011, *Dynamic Adpative Streaming over HTTP - Standard and Desing Principles*

[13] MPEG, 2012-08-26, *MPEG ratifies its draft standard for DASH*

[14] Anthony Vetro, Mitsubishi Electric Research Labs, 2011, *The MPEG-DASH Standard for Multimedia Streaming Over the Internet*

[15] Sodagar I., 2011, The MPEG-DASH Standard for Multimedia Streaming Over the Internet

[16] Edmondo Lopez e Diego Magnani, 4 aprile 2011, *Cos'è Android? La storia del sistema operativo mobile di Google, Android Italy*

[17] Miller, Germany Quacchio, E., Gennari, G. Wolisz, 2012, *Adaptation algorithm for adaptive streaming over HTTP*

[18] Kovacevic, J. Miljkovic, G. Lazic, K. Stankic, 2013, *Evaluation of Adaptive Streaming Algorithms Over HTTP*

[19] Exoplayer Library, 2015, http://developer.android.com

[20] BigBuckBunny video encoding AVC, 2015, http://www-itec.uni-klu.ac.at/ftp/datasets/DASHDataset2014/BigBuckBunny

[21] Lorenzo Keller, Anh Blerim Cici, Hulya Seferoglu, Christina Fragouli, Athina Markopoulou, 2012, *MicroCast: Cooperative Video Streaming on Smartphones*

[22] DashPlayer source ,2015, github.com/x54h

[23] Bitcodin.com, 2015, *MPEG-DASH Overview*

Ringraziamenti

Vorrei ringraziare il mio relatore, Prof. Luciano Bononi, relatore di questa tesi e il mio correlatore, Dr. Luca Bedogni, per la grande disponibilità e cortesia dimostratemi, e per tutto l'aiuto fornito durante la stesura. Un sentito ringraziamento ai miei genitori, che, con il loro incrollabile sostegno morale ed economico, mi hanno permesso di raggiungere questo traguardo. Un ringraziamento particolare alla mia "Chou" per il supporto fornito in questo periodo.... Al mio coinquilino per le notti bianche ed i caffè.... A tutti gli amici, colleghi e tutte le persone care che mi stanno vicino nella vita di tutti i giorni...